(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 January 2001 (04.01.2001)

PCT

(10) International Publication Number
WO 01/01207 A1

(51) International Patent Classification[7]: G05B 17/02

(21) International Application Number: PCT/US00/18179

(22) International Filing Date: 30 June 2000 (30.06.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/346,107 30 June 1999 (30.06.1999) US

(71) Applicant: ETEC SYSTEMS, INC. [US/US]; 26460 Systems, Inc., Hayward, CA 94545 (US).

(72) Inventor: SCHOLTE VAN MAST, Bart, G.; 5945 Corte Espada, Pleasanton, CA 94566 (US).

(74) Agent: KLIVANS, Norman, R.; Skjerven, Morrill, Macpherson, Franklin & Friel LLP, 25 Metro Drive, Suite 700, San Jose, CA 95110 (US).

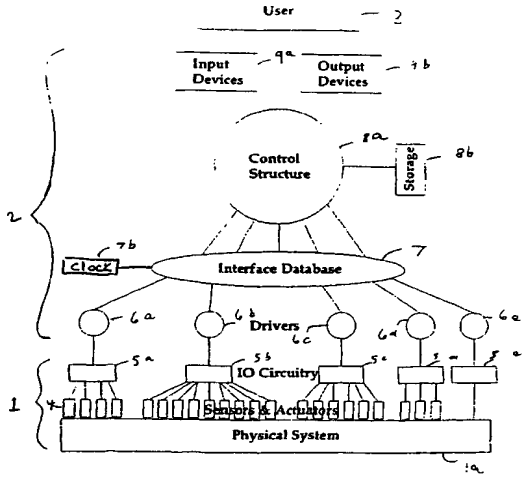
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: METHOD AND APPARATUS FOR HIERARCHICAL CONTROL OF CONTINUOUSLY OPERATING SYSTEMS



(57) Abstract: For simulation and/or control of automated physical ("real") systems, a hierarchical control system is embodied in computer software executable on a general purpose computer. This is suitable for control and monitoring of complex physical systems without use of elaborate interface circuitry. An ergonomic user interface (7) graphically closely adapts to the particular automated system which is being simulated and/or controlled. The automated system is partitioned into various sub-elements, each of which typically represents a physical part, for instance, pumps or gauges or tanks in a chemical processing system. These elements are arranged in a hierarchical structure of elements. Each element has associated with it the intelligence and logic, in terms of the computer software, needed for its corresponding physical functionality.The function of each element depends on its current state as defined by the computer software and the states can be changed by commands as in the physical system.

WO 01/01207 A1

METHOD AND APPARATUS FOR HIERARCHICAL CONTROL OF CONTINUOUSLY OPERATING SYSTEMS

FIELD OF THE INVENTION

This invention relates to control and simulation of continuously operating systems using computer software.

BACKGROUND

Continuously operating systems are common; examples are chemical plants, portions of chemical plants, security systems, automated aircraft, semiconductor wafer processing systems, building HVAC systems, and material handling systems. There are many other examples in both manufacturing industries and other fields. Typically such systems include a number of sensors which sense physical changes, for instance temperature, pressure, speed, flow rates, position, and actuators which control such parameters. The sensors and actuators are coupled to a central controller which typically executes some sort of software. The controller may or may not be a general purpose computer. The controller evaluates the sensor data, adjusts the actuators, and typically provides some sort of a display to a human operator indicating a status of the system. The human operator then may or may not intervene in operation of the system; in many cases the system operates automatically under control of the controller. In these cases the controller is controlling functions of the system.

Alternatively, there are simulations of such systems in which there are no physical sensor inputs or controlled outputs but instead virtual input and output data is provided, for purposes of simulation and/or design of a system. Typically, however, such real or simulated systems require customized software, especially in the simulation regime. Often, systems of this type also require specialized computer hardware. They are thus relatively difficult to program. Often, there is no standardized programming interface but instead custom programs (software). Of course, this increases costs and makes such system relatively difficult to design and implement. Therefore, improvement is needed in such systems for both control and simulation purposes, which typically includes design of such continuously operating systems.

SUMMARY

In accordance with this invention, there is disclosed a generalized method and system for the control and/or simulation of automated (e.g., continuously operating) physical systems.

This control and/or simulation is embodied in computer software executed by a host computer, e.g. a standard personal computer, which controls and monitors complex applications (physical systems) without use of specialized (computer) hardware. Advantageously an intuitive and easy to use graphical user interface adapts closely to the particular physical system being monitored and/or simulated.

The computer software includes a number of elements, each being a computer software object representing a part or plurality of parts of the physical system being controlled and/or simulated. The elements are arranged in a hierarchy, each element being represented by one of a plurality of states. A state of the system is an aggregation of the element states.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 shows a system in accordance with the invention.

Figure 2a shows graphically analog state definitions.

Figure 2b shows graphically an analog state with a "dead band."

Figure 2c shows continuous state transitions.

Figure 3 shows a flow chart for the main application process.

Figure 4 shows a flow chart for the state determination procedure.

Figure 5a shows a flow chart for command interpretation.

Figure 5b shows a flow chart for structural state determination.

Figure 6 shows a flow chart for the structural updating and state calculation.

Figure 7 shows code for the controller state machine.

Figure 8 shows a flow chart for cross link creation.

Figure 9a, 9b, 9c show examples of compiler functions.

Figure 10 shows a tank for holding a liquid in a first example.

Figure 11 shows a chemical dilution system in a second example.

Figure 12 shows a hierarchy for the Figure 11 system.

Figure 13 shows a timing diagram for the Figure 11 system.

Figure 14 shows a timing diagram for a switch of Figure 11.

Figure 15 shows a high vacuum system in a third example.

Figure 16 shows a hierarchy for Figure 15.

The use of the same reference symbols in different drawings indicates similar or identical items.

DETAILED DESCRIPTION

The computer implemented method and apparatus disclosed herein are for the control and simulation of automated (typically continuously operating) physical systems. The present method and apparatus are intended to control, monitor, and simulate complicated physical systems without the use of elaborate dedicated interface or controller circuitry. The method and apparatus provide a user interface that adapts closely to the particular physical system.

In accordance with the invention, a physical (real) system is treated as a combination of interacting physical parts (components). Each part has a software entity counterpart called an element. Each element is expressed (coded) as an object which is the actual software source code used to create an element; hence one object can produce multiple elements. For instance, there is one object which is a generalized binary output, e.g., a real valve in a physical system has a corresponding software valve element which is a particularized version of the output object. The elements are combined (conglomerated) into a hierarchical structure. The function of an element depends on the current state of the element. The method is to assign a state to an element from the combined states of a collection of elements. The method allows fast and reliable state determination.

Each element in the system has an associated graphic (graphical user interface) representation. These graphics can be designed or selected to correspond to the real hardware or schematic diagrams of the physical system. The graphic representation follows the hierarchical structure of the physical system, displaying all the available information on each hierarchical level. This allows the user to browse through the hierarchy.

Figure 1 shows graphically the physical portion of the system 1, the software and computer portion 2 which is executed on a host computer, and the human user 3. The physical portion includes the physical system 1a, the sensors and actuator 4, and the I/O (input/output) circuitry 5a, ..., 5e. The software and computer portion 2 includes the device drivers 6a, etc., the interface database 7 and its associated clock (timer) 7b, the control structure 8a which accesses associated storage (e.g., hard disk) 8b, and the input devices (e.g., mouse, keyboard) 9a, and output devices (e.g., computer screen) 9b with which user 3 interacts.

In one embodiment, the software and computer portion 2 is executed on multiple host computers linked into a network rather than a single computer. Even execution of the control structure 8a may be distributed over several networked host computers. The user 3 access may be by a remote computer through a network. Figure 1 will be better understood from the following disclosure.

Control Logic

The software of control structure 8a is organized in a hierarchy of objects of different types. The fundamental object types are used to create the control structure elements where the same coded objects are used multiple times to represent multiple elements. With these elements, the structural functionality of the physical system 1a is described. The following object types are present in the control structure 8a:

Basic Elements

The entire control structure is based on a few basic element types. Each object in the control structure is derived from these basic types.

Element - Each item defined in this portion of this disclosure is referred to as an element. An element is any kind of software object that is defined in the system. Elements have different forms and functions depending on their definition. Elements can be stored and retrieved in storage 8b. Elements have a graphical representation on the output device (screen) 9b.

List - All the elements are arranged in lists of multiple elements. Lists can be displayed, modified, stored and restored. Lists are dynamic structures without predefined length, format, or dimension.

Graphical element - A graphical element holds the coordinates and the form of any graphic representation. These representations are points, lines, boxes, circles, text, icons, etc. Graphical elements are displayed on the screen with respect to a window as described below.

Icon - A list of graphical elements is referred to as an icon.

Icon library - The graphical library is a list of icons. In this list, each icon is defined only once. Since in the list that makes an icon, references to icons within the icon library are allowed, the graphic library is an internally linked, annotated structure.

Window - A set of translation vectors, scale factors and other image transformations. Windows are used to display each icon or element in its screen representation. A window could shift the element on the screen and can scale the element representation to the current environment. The transformation, applied by the window, is also applied to the windows in the transformed icon. This allows the use of icons within icons to infinite levels. An icon within an icon is drawn using repeatedly transformed coordinates.

Screen - A screen contains a list of items that are currently visible on the screen. The screen will update an item on the screen every time its representation changes.

Hierarchical Elements

The elements defined in this subset are the basic structural units and, when combined, form the hierarchical representation of the control structure. Elements combined in a structure are referred to as nodes.

Node - A hierarchical node is an element with one connection to higher levels and multiple connections to lower levels. Downside connections are arranged in a list. The number of levels is not limited. The control structure 8a is made by arranging a number of nodes. Nodes can be defined to perform tasks as function of their present state. In this form, nodes act as subsystems. To identify nodes of any type in the control structure, each node has a unique identification label.

Child node - A child node is a node seen looking down the hierarchy.

Parent node - A parent node is a node seen looking up the hierarchy.

Endnode - An endnode is a hierarchical node with only one connection to higher levels and no connection to lower levels. Endnodes are nodes at the lowest end of a structural branch. Several types of endnodes can be predefined. Each endnode definition gives the endnode certain specific functionality. Endnodes are used to make connections to the hardware of the system as IO (input/output) devices but also have other functions. Endnodes perform tasks in the same way that nodes can but the number of states of an endnode is typically lower.

Cross link - Cross links are direct, one directional links between two nodes. Since a link can be made with any node, it can be used to bypass the limitations of a hierarchy. After a cross link is established, all the information of the node at the end of the links is available to the node at the beginning. Cross links are also used by the function compiler as defined hereafter. Cross links are made as needed. Established links can be maintained after use.

Guest node - When a node is linked to other nodes by a cross link, this node is called the guest node of the element to which it is linked.

Hostnode - The node that carries the link. Specific types of hostnodes can be defined.

Shadownode - A shadownode is a hostnode with no other function than to be a transparent representation of another node or endnode in the structure, its guest node. Shadownodes can be used to link an entire branch of the control structure to another, without redefining this branch. A shadownode of an endnode is useful to make certain information immediately available throughout the structure. Commands, as defined hereafter, sent to a

shadownode are directed towards the guest node. If no cross link is available, commands to this node are ignored.

Dynamic node - A dynamic node is a shadownode without a pre-defined destination of its cross link. If a cross link is established, the node acts as a normal shadownode. Establishing the cross link for a dynamic node is initiated by sending it a predefined command that carries the identification label of the desired guest node. It will take the identification label and start the normal cross linking procedure as defined hereafter with this label as defined for the shadow node. This implicates that if the node has an established cross link, the new link will not be formed. Terminating the cross link is done by predefined command.

Access Node - An access node is a dynamic node that has a connection to an interface device.driver 6a, etc., either via the interface database 7 or directly. All information available to the access node can be transmitted over the communication line by sending a predefined command to this node. Specific parameters can be selected by adding the parameter identification to the command (e.g., ParID = 1: node state). If the cross link is not established, the transmitted data is taken from the access node itself. All information entering the node through the communication line is posted as internal commands. This automatically allows the access node to create and terminate cross links throughout the control structure. The entire control structure can therefore be addressed through the communication line. Communication protocol and error checking are defined in the communication interface device driver.

Active Elements

Hierarchical nodes define the static framework of the control structure, and active elements are used to cause action in the control structure. They make direct status-related communication between hierarchically linked nodes and to make nodes perform certain tasks. They are also used to create a command-action sequence possible of non-hierarchically linked nodes.

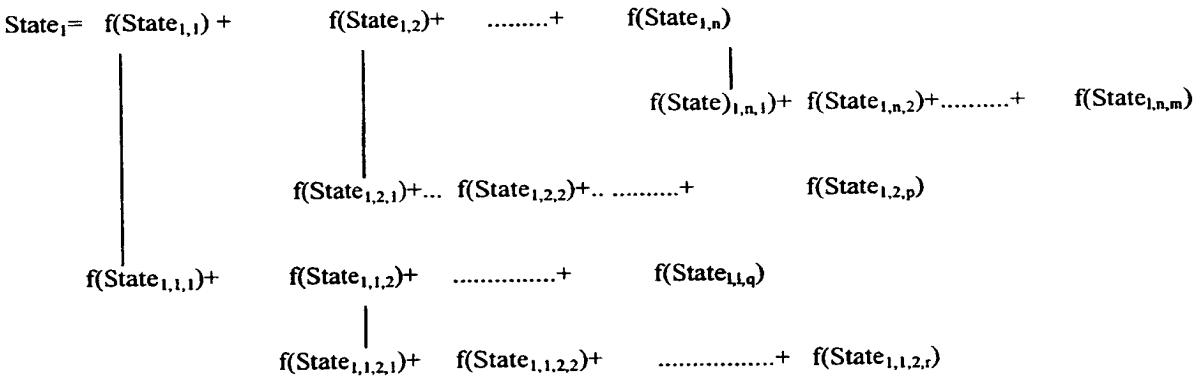
State List - All the active elements are arranged in lists of multiple active elements. Every node carries a state list. These lists are available to the nodes to select relevant information.

States - States are active elements that define the current situation and function of a node. Every state has at least one state code. The state code is a numerical representation of the state and defines in what state the node will be in a particular situation. Each state code should only be addressed once. Multiple states with identical codes lead to unpredictable state selections.

Each state has one update code. The update code is therefore dependent on the state of the code. The update code is used to transmit the present state of this node to the hierarchical level above. With these two parameters the actual situation of the node is determined. For endnodes, the determination function of the state is device-specific. For nodes (or subsystems) the state is in one embodiment calculated by the expression:

$$State = \sum_{k=1}^{m} Update_k(State_k)$$

in which k is the number of the child node and m is the total number of attached nodes. Since $state_k$, the state of childnode "k", is calculated with the same expression, the complete structure is updated by calculating the state of the highest level node, of which there is only one. The actual calculation can be represented as:



in which f is the relation between state and update codes, and n, m, p, q, r are indices. Both the state and update code range in value (in one embodiment) from 0 to 255.

The above definitions are mostly logical AND operations. To make it possible to implement logical OR relationships, each state can have more than one state code. The state is appointed when the state calculation leads to a state code in this list. Another way to obtain an OR relation is to define multiple states with the same update code. The OR relation is then established one level higher in the hierarchy.

In another embodiment, instead of the state aggregation being a sum of the states, it is a product (multiplication) of the states, or a combination of a sum and a product.

To accommodate the human user 3, each state has associated displayed text and has a specific effect on the representation of the nodes. In one embodiment, the node is displayed with the current state's name, and every child node is displayed in a state color of the child node.

State Types

Passive state - If a state is only used to transmit information inside the hierarchy through its update code, this state is passive.

Active state - An active state transmits at least one command, as defined hereafter, directly after it has been selected. This state carries its own list of commands. All the commands in this list are transmitted at once. It continues transmitting until the state is aborted and another state becomes current.

Not-Active state - This state is similar to the active state, but it transmits its commands when it is aborted.

Delayed state - A timed delay postpones the transmission of its command list until a certain time has expired. The timer starts when the state becomes active. When the state is aborted before this time was reached, no actions are taken.

Timed state - A timed state postpones the transmit of its command list until a real time clock of the host computer indicates a certain time and/or date. When the state is aborted before this time was reached, no actions are taken.

Wait state - A wait state utilizes the possibility to create cross links throughout the control structure. It only transmits the command list after the node at the end of its cross link reaches a certain state. If the state is aborted before this happens, no action is taken. The first time the state is active it establishes the necessary connection.

Conditional state - The conditional state also uses a cross link connection to another node. This state has two associated command lists. Which command list is transmitted depends on the state of the node at the end of the cross link. Conditional states act immediately after the successful creation of the cross link.

Commands

Commands force nodes into a state. Commands are transmitted by active states of any type and can be sent to any node in the structure. The destination of a command is declared by entering the identification label, e.g., number of the destination node. The actual command is the code of the state that this node should activate. The node will select this state immediately, without checking its IO or calculated/real-time state. The node performs all the actions linked to this state. Commands usually only have state information but can carry additional parameters when this is needed. These parameters can be any kind of number, e.g., analog set points but can also be structural information.

Command list - A list of commands that is linked to a certain state or element. Command lists are transmitted as one unit, but each item of the command list can be retrieved individually.

Command stack - To enable the commands to be transmitted throughout the structure, all commands that are transmitted are stored on a central command stack. This stack is scanned by every node before it makes its update calculation. If a node finds a command with its identification number, it is put in this state immediately. The command is then removed from the command stack. It is preferred to allow nodes to send commands mainly to their own child nodes. This keeps the stack low.

Reports - Reports are shadow-representations of a node. A report that a node transmits contain the nodes identification code and the structural location of this node. Transmitting a report can be forced by sending a predefined command to a node. Reports are used to form cross links between nodes.

Report stack - To ensure that reports are available throughout the structure, all reports that are initiated are stored on a central report stack. This stack is scanned by every node that is trying to establish a crosslink. If a node finds a report with the desired identification number, the structural information in the report is used for the creation of a cross link. The report is then removed from the report stack.

Control elements

With the elements defined above, it is possible to build an active hierarchical structure. The basic elements of these structures are nodes and child nodes with states. These elements are control elements.

Controllers- Controllers are nodes that contain all the information in their states to perform a certain task. Controllers can be displayed as an icon with a sub-icon for each sub-controller it manages. Controllers are completely user defined and have no intrinsic intelligence.

Systems - A system is a controller like any other, except that it displays its descendants as controllers. The descendants are updated continuously with their full representation. This enables the user to monitor several controllers simultaneously.

Devices

The intended purpose of the control structure is to manage a large number of devices in an organized way. These devices are usually endnodes with all the necessary states. Since devices are the software link to the physical (real) world, they include a number of features

that enables them to perform a certain task. In most case, this means that a few of the, e.g., 256 (0 to 255) states that a device can have are predefined. State transitions are then driven by events in the physical portion 1 through interface device drivers 6a, etc.

Since device state transitions are event driven, it is possible to put a device in a different state than is the actual representation of the interface by a command. After a physical event, the device representation will again be in accordance with the physical interface.

Transitional devices - A transitional device is only capable of making state transitions as a result of commands. These devices are useful to set controllers into certain states, without having to change the actual hardware platform. This enables the user to let the system react differently to identical situations.

IO devices - IO (input/output) devices are devices that represent the value of items in the interface database in a few predefined states. They may address these items individually. The types of IO devices are as follows.

Digital input devices - Digital input devices have two predefined states corresponding to their input signal. Logical low will yield state code "0", logical high will result in state "1". The source of the digital signal is determined with a parameter set "channel" and a "line" parameter. "Channel" defines the slot and "line" appoints the bit on this slot of an item in, for example, the interface database.

Digital output devices - Digital output devices have two predefined command states that involve switching an item in the interface database. This is similar to the digital input.

Analog input devices - Analog input devices represent an analog input voltage (signal). This voltage can be scaled using a scale function. Analog input devices may also contain a setpoint and a maximum relative tolerance. With the actual analog input signal these numbers determine the state of the device. The predefined states are declared as represented graphically (in an example) in Figure 2a or by any transfer function. Figures 2b and c show particular rounding function to determine a state value from an analog deviation value. The signal input source is defined by the "channel" parameter. This number represents the source of the analog signal.

Analog output devices - Analog output devices have several predefined commands, all carrying parameters. These commands are used to set, increase and decrease the analog output value of the device. The state of an analog output device is determined similar to that for the analog input.

Linked devices - Linked devices utilize the cross linking possibly. A linked device processes the data of one or more other nodes.

Arithmetic devices - Arithmetic devices have an extra field where a string representing any calculation can be entered. The function is compiled (see description of function compiler below), and is used in a compiled format during runtime. The result of the calculation is the current value of the device. State determination is performed as for an analog device.

Change devices - Change devices monitor absolute change of the value of a guest node. The device is reset at a certain state and it stores the value of the guest node at that time. The device continuously monitors the difference between the stored and the present result of the guest node. The state determination is the same as for an analog device.

Derivative devices - Derivative devices are normally linked to an analog device. It accepts the actual value of the analog device and differentiates it with respect to time. The number of data samples that the device uses is variable. The actual derivation uses the following linear regression approximation of the curve slope in time:

$$Slope_{(t_n=t_o)} = \frac{n \cdot \Sigma value_n \cdot t_n - \Sigma value_n \cdot \Sigma t_n}{n \cdot \Sigma t_n^2 - (\Sigma t_n)^2}$$

Where n is the number of data points, t is the time since t=0 and "value" is the actual value of the analog device. The device state is defined as for analog devices. See the graph of Figure 2c for an example.

Integrating devices - Integrating devices calculate its present result using the following function:

$$Integral_t = integral_{t-dt} + Value_t \cdot dt$$

The state determination of this device is identical to the state determination of an analog device.

System devices - System devices are devices that have special capabilities with respect to the user or system. These devices are usually displayed as a descendant of the top-level system controller.

Clock devices - Clock devices display the current value of a real time clock. The value is updated whenever a state change occurs.

Speed devices - Speed devices display the number of times it has been updated since the last state change.

Memory devices - Memory devices read the system memory. This device is especially useful during system development. The total memory usage of the system structure can be monitored. It is also possible to see if any unaddressed commands are transmitted. This leads to a continuous decreasing of available memory. This device is updated when a state change occurs.

Message - Message devices keep track of all the messages that are sent from the hierarchical structure during runtime. The state descriptions of this controller are the actual message text. It is still possible to connect an action to a certain message. The difference is that these commands are only transmitted once. With the message text, the accompanying command name (usually an indication of the source of the message) and the time of receiving the message command is displayed.

All messages are stored in a text file for later backtracking. The message file also contains the date of the message.

Manual commands

Entering of commands occurs in two different ways. Both are linked to a node and are only available in that node. The given commands however, are treated as if it where internal system commands send by states. Thus, manual commands are not limited to the node that carries the command item.

Command menu - This menu is hidden under a representation displayed on the node whenever the list is not empty. The list is opened when the user selects this button. The commands in the list are defined during configuration of the system. The user can select a command from this list. The selected command will be put on the command stack immediately. A node can carry more then one command list.

Command buttons - A command button is displayed on the node as a single object. This button represents a command list. When the user presses the command button, the complete command list is placed on the command stack. The commands in the list are defined in the configuration. A node can carry more then one command button.

Interface Elements

An illustrative example of an interface platform (see Figure 1) used in the present system is a commercially available National Instruments SCXI-based chassis. The actual software link from the host computer executing control structure 8a to the chassis is via the hardware device drivers 6a, etc., (which themselves are software but are hardware dependent). Device drivers 6a, etc., continuously update the software representation of the interface chassis

in the interface database 7. This software representation depends on the configuration of the chassis.

Chassis - The chassis is the collection of IO circuits 5a, etc., (also referred to as "boards" for circuit boards as described hereinafter).

Analog input board - The analog input board device driver manages, e.g., 32 channels of analog input as presented by an analog multiplexer board. The channels are periodically scanned during runtime of the system. Each channel can be sampled several times before the average value is written in the interface database. To avoid cross channel overlap, the driver switches to the next channel after measuring but then leaves the analog measuring procedure. During the next analog update cycle the then selected channel is updated.

Analog output board - Complementary to the analog input board.

Digital input board - The digital input board driver updates all, e.g., 32 bits of the digital input database to agree with the bitmap of the corresponding physical inputs during each updating cycle.

Digital output board - The output board driver updates all, e.g., 16 channels of the digital output. The lines of the board are set as the bits are present in the digital output database for this board.

Serial IO board - This IO circuitry supports serial communication.

Implementation Methods

The basic types and functions of the elements in the software system are described above. The following describes the interaction between these elements, including processes ("procedures") used by and with the elements to perform their tasks.

Main Method - The main runtime cycle of the software system includes several procedures (see Figure 3). The first procedure 10 is the hardware interfacing update. Next the control structure is updated at 12 to reflect the current hardware situation. If there is a user input device action at 14, - e.g., a key or a mouse button is pressed on the host computer - the user input is handled. Depending on whether the menu is active or not at 18 the user is allowed to perform different actions. If the menu is not active, the user can only browse through the structure or enter manual commands into the structure at 20. If the menu is active, at 24 the runtime editor is active and will interpret the user action. In this mode, the parameters can be altered. If the screen representation of visible elements has changed at 25, these representations are updated at 26.

Commands and state transitions

State selection - A state is selected by entering the state selection procedure by inputting a specific state code at 30 (see Figure 4) which illustrates this state determination procedure. This code is a command number or a determined state. In the state selection procedure, the state list that accompanies the current node is searched at 32 for the state with the entered identification code in its code list. If the state code is found at 34 ("yes"), the state is selected. The present state of the device is then canceled by the abort sequence 38 determined in this state. After that, the new state is initialized at 40.

If a state code is entered in the state determination procedure that is not found at 32, the state determination procedure restarts its search at 36b with a default error state code. This state code is then set at 36, after which the state list is searched again. If the error state is not found, the procedure creates this state at 36c. The state is added to the state list of the device as a passive state with a pre-defined update code.

Command Interpretation - The first step that each device takes in its update cycle is the scanning of the command stack for any incoming commands, as shown in Figure 5a illustrating the command interpretation process. The scanning of the command stack at 44 is abandoned whenever a command is found at 46 or at the end of the stack. This means that during one updating cycle only one command is taken. If there is more then one command for one specific device, they are handled in subsequent updating cycles. If a command is found at 46, the command code number is taken as the code to enter the state selection procedure at 48 followed by a current state update at 48b. The state selection (determination) process 48 is shown in detail in Figure 4. The state that is identified by the code is activated immediately.

Structural state determination - The structural state determination function, as defined above, allows rapid and explicit state determinations. To make a state determination, a node asks all its child nodes to report their update codes, e.g., calculate their states. The node takes the total sum of these codes. This summation is the number that the node uses to enter the state selection procedure. The node enters this procedure only when the result of the summation differs from the state code of the current state. This process is illustrated in Figure 5b. The state calculation is initialized at 45 to enter a state calculation at 47. Child nodes are checked at 49 and updated at 51 to report the update result at 53 to 47. Only when all child nodes are exhausted at 49 is this process exited at 53. Subroutine 51 is the identical process for the childnode. This process descends into the hierarchy until the end of the branch is reached, i.e. an endnode is encountered.

Structural updating - The procedures described above are all that is needed for the structural updating and state calculation as shown in Figure 6 beginning at 50. First, each node starts with the command scanning and interpretation at 52 (shown in detail in Figure 5c). Then the actual node state determination is executed at 54 (shown in detail in Figure 5b). State selections 58 (shown in detail in Figure 4) are only performed when needed as determined at 56. This means that the device will only scans the state list whenever the lower levels change, or when a command is entered. In each cycle, the state update is performed at 64. This is done to allow states that have running timers to update their timers or conditional states to check conditions. An example of code (software) for structural update of a node (the controller state machine) is shown in Figure 7 in the Pascal language.

Cross link creation - A cross link creation process (see Figure 8) is initiated at 72 by the element that needs the link. If the link pointer is empty at 74, the element scans the report stack at 78 for the representation of the device it wants to link with. If the element does not find the information it needs, it transmits a pre-defined command at 80 that forces the potential guest node to put its structural information on the report stack. The element then leaves the current linking cycle at 83.

The potential guest node now finds a link request command on the command stack and transmits its coordinates to the report stack. This is the only action that the guest node takes. During the next cycle, the element that needs the link finds the information of its guest node on the report stack. This information is removed from the report stack and stored in the link pointer at 82. After this cross-link creation, all information of the guest node is available to the linking element.

Interfacing - The actual control system is indifferent with respect to the hardware platform (chassis) it controls. To accomplish this, a hardware device driver 6a, etc. (Figure 1) (software) is in the main runtime loop. This device driver updates the interface database 7 of the hardware situation each time it is addressed. It also sends out any changes to the hardware settings made by the control structure 8a. Immediately after this cycle, the physical portion 1 is exactly represented in the database 7. The corresponding item in the interface database 7 is updated in real time. The control structure 8a only uses the software representation by direct reading and writing on the elements of the database 7.

Function compiler - Any number in the control structure 8a can be replaced by a function. In addition to the standard arithmetic functionality, the function compiler can interpret a number of other operations. These operations or functional relations are used to

speed up evaluations. The functional dependence of state parameters may be established by the function compiler. The function compiler may directly influence a state of a node, e.g., the compiled function may be a component in a child node list of a node.

Standard functionality - Examples of standard arithmetic operators in the compiler are shown in Figure 9a, where 'a' and 'b' are numbers or other functions. Examples of standard mathematical functions in the compiler are shown in Figure 9b.

Boolean functionality - Boolean operators in the compiler are shown in Figure 9c. The result of these evaluations is a real number with value '0' if false and '1' if true. The result of the '~' operator depends on the setting of a range variable. This evaluation is used to determine whether two real values (a and b) do not deviate more then a certain amount. The actual evaluation algorithm is:

$$|\frac{a}{b} - 1| < \Delta$$

in which Δ is the range setting.

Relative functionality. There are two ways of relative addressing of numbers. The first has the form:

[*DevID*,*Par*ID]

This function returns the current value of the parameter of the device for the identification numbers entered between the brackets. The function creates the cross link when it is updated and the link has not been established yet.

The other form is used to directly address the interface database 7. The actual form depends partly on the format of the elements in database 7, but the general form is:

IOP (ChassisID, SlotID, LineID)

The result of this function is the current input or output value of the addressed item in the database 7. Both methods can be reversed to write new values in the addressed locations.

Editing

The runtime editor is a special re-entering editor whose functions can be inserted in the main cycle. The editing functions change parameters and settings without interrupting the control sequence. The editor also allows the user to move items over the screen to any desired position. It is possible to instruct the system to open more then one viewport to the control structure. This allows the user to have multiple controllers on one screen. All items on the screen can be edited by the editor. Whenever the system is halted, the identification codes of the items on the screen are stored in the default screen item list.

No structural changes are allowed during runtime. This means that no subnodes can be removed or added, and that no node identifications can be changed. However, it is possible to alter the state and command tables of the nodes. The runtime editor is only active when the menu structure is active.

In addition to the functionality of the runtime editor, the structural editor incorporates the structural modification functions. With this editor the system structure is defined or modified. It allows the creation, deletion and modification of structural branches and nodes.

Operational Modes

There are three operational modes:

Normal - The normal mode runs the system as quickly as possible. This mode should be used after the system performs as intended.

Step - The stepping mode allows the user to monitor the system evaluation while it takes place. The updating sequence of each node in the structure is halted until a key is pressed. Then the next node is updated and will be displayed on the screen with all current data.

Learning - In learning mode the system runs normally, until it encounters an undefined state. In normal mode, the default error state is selected whenever this happens. In learn mode the system interrupts its updating sequence. The node that has entered the error state is displayed. A small submenu asks the user if this state should be defined, assigned to an other state or ignored.

Defining the current state enters the normal state definition sequence. Selecting an already defined state will add the undefined state code to the state code list of the defined states. Selecting ignore will no longer cause the system to halt at this node for this state. After completion of the state definition, the system continues running with the new state.

The system does not store the modifications; this is done manually. Clearing the learn mode by selecting another running mode, also clears all the ignore flags.

First Example: Liquid Tank

The basic method described above is elucidated by a simple example. In this example there is a liquid container (tank 72), in which two sensors 74a, 74b monitor the liquid level. One low-level sensor 74b is at the bottom of the tank 72 and a high level sensor 74a is at the top (see Figure 10). Several states have to be defined for tank 72. One wants to be able to detect if the tank is "full", "empty" or somewhere in between. This last state is defined as "ready".

Both sensors 74a, 74b can have two states, "wet" or "dry". These two states are directly related to the digital input ("high" or "low") from the sensor. With these states one defines the following state table for the tank:

Tank	Low Level		High Level		State
	State	Update	State	Update	
Empty	Dry	0	Dry	0	0
Ready	Wet	1	Dry	0	1
Full	Wet	1	Wet	2	3
Error	-	-	-	-	255

The table starts with the assignment of the "Empty" state. Both sensors are "dry". Update code "0" is assigned t both states. If the tank is slowly filled, the lowest sensor switches to "wet". This state is assigned an update code "1" leading to a unique state "Ready" in the state calculation. If the tank is filled further, the high level sensor switches to "wet". Update code "2" is assigned to this state. In summation, this leads to a "Full" state with code "3".

It is possible to assign updated code "1" to the high level sensor's "wet" state, as was done for the low level sensor. However, with the assignment of update "2", an undefined state is inserted for the tank, namely "2". If in the current table, state code "2" is encountered, this would mean that the low level sensor is "dry" and the high level sensor is "wet". Since this is physically impossible, this indicates a defective sensor. If state update code "1" would have been assigned, this discrimination between sensors would not be possible.

This is a simple application of the method which shows that, by properly defining the update functions, unique states are identifiable. Leaving states undefined can serve as an error detection mechanism.

Second Example: Chemical Dilution System

A second and more complex example is a chemical dilution system. The physical system controls the dilution of concentrated chemical with water to the desired concentration and as shown in Figure 11 has two major parts. The first is the batch controller 122 which is, e.g., a programmable pulse divider. The controller 122 (a physical part) receives high frequency digital pulses from the flow meter 124. It divides the number of pulses to a level that is send to the metering pump. Both the batch controller 122 and the metering pump 128

are monitored for proper action. The physical system also has a chemical storage tank connected to inlet 130 from which it takes its chemicals. This tank, and all the sensors and systems around it, including regulator 134 and water valve 132, are controlled.

The corresponding software (control structure) hierarchy is depicted in Figure 12 for the Figure 11 physical system. All the elements are accompanied by their corresponding device identification codes.

The following shows the logical tables that control the diluting system of Figure 11. Tables for the tank and for an external supply unit to fill the tank are omitted for brevity.

a. Metering Pump, Control Pulse. The steering pulse from the batch controller 122 to the metering pump 128 is monitored on a digital input line. Because it is only necessary to know if the pump 128 is running, the actual digital state of the pulse is not important. Therefore, this pulse is converted to a two-state update code that identifies the running state of the pump. This is done by adding a third state to the controller. This state is always made active if any of the other two is active for more then a certain time. These other two states are the default digital input states. Both digital states have the same update, so they are indistinguishable for the higher level controller. The timing diagram for controller 122 is in Figure 13. The corresponding state transition table is:

State Name	Input		Action		State	Update
Low Stroke	Low	0	t	go OFF	0	2
High Stroke	High	1	t	go OFF	1	2
OFF		0/1			2	0

b. Metering Pump. The default state of the metering pump 128 is "OFF". (See the following table.) In this state the pump 128 is disabled and does not start pumping if control pulses are available. Before starting, the pump 128 is set in the "READY" state by closing the "pump enable" relay. If control pulses are available, the pump controller goes "RUNNING". If the internal alarm relay of the pump 128 is released, the pump is set to "OFF-error". If this happens during ready or running states, the pump 128 is disabled.

State Name	Pump enable		control pulse		alarm relay		Action	State	Update
OFF	OFF	0	OFF	0	OFF	0		0	0
READY	ON	1	OFF	0	OFF	0		1	1
RUNNING	ON	1	STROKE	2	OFF	0		3	4
OFF-error	OFF	0	OFF	0	ON	4		4	10
ERROR							MP dis	255	

c. Batch controller Start / Stop relays. The batch controller 122 is controlled by two remote switches. These switches start or stop the controller 122 when pressed for a short time. The corresponding digital output devices are set to fall back to their default position, some time after activation. An external command is the only way that these devices can be activated. The corresponding timing diagram is given in Figure 14. The delay time for the two switches is set to different length. The starting pulse is very short. The stopping pulse is quite long. The state transition table is:

State Name	Input		Action		State	Update
OFF	Low	0			0	0
ON	High	1	t	go OFF	1	start=1 stop=4

d. Batch Controller. The batch controller 122 has remote switches, a steering relay for the DI-water valve 132 and an overrun alarm relay. It only handles the "Running" state as active. The total time in this state is limited. The overrun relay sends out commands independently. It shuts down the system and fires the major alarm when activated. The state transitions are:

State Name	Start Relays		Stop Relays		Valve Relays		Overrun		Action		State	Update
STANDBY	OFF	0	OFF	0	OFF	0	OFF	0			0	0
starting	ON	1	OFF	0	OFF	0	OFF	0			1	0
started	ON	1	OFF	0	ON	2	OFF	0			3	2
Running	OFF	0	OFF	0	ON	2	OFF	0	t	go STOP	2	2
stopping	OFF	0	ON	4	ON	2	OFF	0			6	2
stopped	OFF	0	ON	4	OFF	0	OFF	0			4	0
Command States												
START	Start relays ON										100	
STOP	Stop relays ON										101	

e. Acid Mix System. This controller manages the batch controller 122 and its metering pump 128 as shown in the following table. It is a combination of the batch controller and its metering pump:

State Name	Metering Pump		Batch Controller		Action		State	Update
OFF	OFF	0	Standby	0			0	0
STANDBY	Ready	1	Stopped Standby Starting	0			1	0
starting	Ready	1	Started Running stopping	2	t	STOP	3	2
Running	Pumping	0	Started Running Stopping	2	t	STOP	2	2
stopping	Pumping	0	Stopped Standby Starting	0	t	Alarm 2	6	2
ERROR					t	Stop Alarm 1	255	0
Command States								
STANDBY	Metering Pump Enable						99	
START	Batch controller start						100	
STOP	Batch Controller Stop						101	
OFF	Metering Pump Disable						102	

f. Chemical Blending System. This controller (not shown) lets the complete blending system, including its tank, line and supply, act as one unit. The default state of the controller is Standby as shown in the following table. In this state the system is on line and the mixing system is off. This means that the metering pump 128 is disabled. After a start request, the metering pump 128 is enabled which sets the mixing system standby. The system is now Idle. This idle state has its own timer that puts the mixing system running by starting the batch controller 122.

State Name	Concentrate Tank		External Supply		Acid Mixing System		System Line		action		St	U
OFF empty	empty	0	off	0	off	0	off	0			0	0
OFF Loaded	low, ready, full	1	off	0	off	0	off	0			1	0
Standby	l, r, f	1	off	0	Off	0	on	7			8	0
Idle	l, r, f	1	off	0	Standby	2	on	7	t	go run	10	57
Running	l,r,f	1	off	0	running	4	on	7			12	63
Filling init	l,r,f	1	Press.	57	off	0	off	0			58	0
Filling start	l,r,f	1	filling	63	off	0	off	0	t	stop fill warning	64	0
Filling	l,r,f-inc	69	filling	63	off	0	off	0	t	stop fill warning	132	0
Filling done	l,r,f-inc	69	off	0	off	0	off	0			69	0
ERROR									t	off Warning	255	
start									a	AMS Stb		
stop									a	AMS off		
ON									a	Line On	251	
OFF									a	AMS Off Fill Off Line Off	252	
Tank Fill									a	AMS Off Line Off Fill Start	253	

Third Example: High Vacuum System

A third example is a high vacuum system, common in high vacuum technology. Although much of the functionality of the physical system is normally in the actual hardware, in this example the control system controls each part. The (physical) high vacuum system of Figure 15 consists of a vacuum chamber 90 that is pumped with a high vacuum cryo pump 92 connected to a forepump 94. A number of valves 96, 98, 100, 102, 110 are used for the

operation of this physical system and pressure measurements 104, 106, 108 are installed to allow monitoring. The physical system is partitioned along the dotted lines in Figure 15.

The dotted lines in the physical system overview of Figure 15 are boundaries of hierarchically separated portions of the corresponding software. The corresponding software (control structure) hierarchy is depicted in Figure 16. The function of each element can be displayed in logical tables. The following tables contain the elements' states as well as the sub-elements states and update codes:

Cryo	Temperature		Line		State	Update
	State	Code	State	Code		
Off	Off	0	Off	0	0	0
Cooling	Off	0	On	1	1	
	Busy	2	On	1	3	4
OK	OK	4	On	1	5	12
Off-busy	OK	4	Off	0	4	
	Busy	2	Off	0	2	41

HiVac	Cryo		Foreline Valve		Cryo Pressure		Nitrogen Inlet		State	Action	
	State	#	State	#	State	#	State	#		@	Description
Off	Off	0	Close	0	HiPress	0	Close	0	0		
Evac	Off	0	Open	1	HiPress	0	Close	0	1	T	ERROR
Evac-done	Off	0	Open	1	LoVac	2	Close	0	3	A	Close V_{fl} Cryo on
Cooling	Busy	4	Close	0	LoVac	2	Close	0	6	T	ERROR
	Busy	4	Close	0	HiVac	7	Close	0	11		
Run	Ok	12	Close	0	HiVac	7	Close	0	19		
Run-error	Ok	12	Close	0	LoVac	2	Close	0	14	A	Cryo off Open V_{fl}
	Ok	12	Close	0	HiPress	0	Close	0	12		
Regen 1	Busy	41	Close	0	HiVac	7	Open	20	68		
	Busy	41	Close	0	LoVac	2	Open	20	63		
Regen 2	Busy	41	Close	0	HiPress	0	Open	20	61	A	Close N_2 Open V_{fl}
Regen 3	Busy	41	Open	1	HiPress	0	Close	0	42		
Regen 4	Busy	41	Open	1	LoVac	2	Close	0	44	T	Open N_2 Close V_{fl}
	Busy	41	Open	1	HiVac	7	Close	0	49		
Go on										A	Open foreline
Go off										A	Cryo off Open N_2

Foreline	Pressure		Pump		State	Update
	State	Code	State	Code		
Off	HiPress	0	Off	0	0	0
Pump start	HiPress	0	On	1	1	
	Vac 1	2	On	1	3	2
OK	Evac	4	On	1	5	4
Off-busy	Vac 1	2	Off	0	2	
	Evac	4	Off	0	4	2

Chamber	Pressure		Bypass valve		Highvac valve		Vent valve		State	Action	
	State	#	State	#	State	#	State	#		@	Description
Vent	Atm	0	Close	0	Close	0	Close	0	0		
Pump Forevac	atm	0	Open	1	Close	0	Close	0	1		
	Busy	2	Open	1	Close	0	Close	0	3		
Forevac done	Vacuum	4	Open	1	Close	0	Close	0	5	T	Close Bypass
Vacuum	Vacuum	4	Close	0	Open	6	Close	0	10		
Vac_error	Busy	2	Close	0	Open	6	Close	0	8	A	Close highvac
	atm	0	Close	0	Open	6	Close	0	6		
Go Forevac	Busy	2	Close	0	Close	0	Close	0	2	A	Open Bypass
Forevac	Vacuum	4	Close	0	Close	0	Close	0	4		
Vent busy	Busy	2	Close	0	Close	0	Open	12	14		
	Vacuum	4	Close	0	Close	0	Open	12	16		
Vent done	atm	0	Close	0	Close	0	Open	12	12	T	Close vent
Go On										A	Open Bypass Close Highvac Close Vent
Go vacuum										A	Open Highvac
Go vent										A	Close Highvac Close Bypass Open Vent

Vacuum System	Switch		Foreline		Highvac		Chamber		State	Action	
	State	#	State	#	State	#	State	#		@	Description
Off	Off	0	Off	0	Off	0	Vent	0	0		
Vacuum init	Vacuum	1	Off	0	Off	0	Vent	0	1	A	Foreline on
Pump busy	Vacuum	1	Busy	2	Off	0	Vent	0	3		
Pump done	Vacuum	1	On	4	Off	0	Vent	0	5	A	Highvac on
Cryo Evacuate	Vacuum	1	On	4	Busy	6	Vent	0	11		
Wait Cryo	Vacuum	1	On	4	Cooling	12	Vent	0	17	A	Chamber on
Vacuum busy	Vacuum	1	On	4	Cooling	12	Busy	18	35		
	Vacuum	1	On	4	Cooling	12	Forevac	36	53		
	Vacuum	1	On	4	OK	54	Busy	15	74		
Evacuate Init	Vacuum	1	On	4	OK	54	Vent	0	59	A	Chamber on
Vacuum done	Vacuum	1	On	4	OK	54	Forevac	36	95	A	Chamber vacuum
Vacuum	Vacuum	1	On	4	OK	54	Vacuum	96	155		
Vent Init	Vent	156	On	4	OK	54	Vacuum	96	310	A	Chamber vent
Vent busy	Vent	156	On	4	OK	54	busy	15	229		
Vent	Vent	156	On	4	OK	54	Vent	0	214		
Off Init	Off	0	On	4	OK	54	Vent	0	58	A	Cryo Off Chamber Vent
	Off	0	On	4	OK	54	Busy	15	73		
	Off	0	On	4	OK	54	Vacuum	96	154		
Off Cryo	Off	0	On	4	Busy	6	Busy	15	25		
	Off	0	On	4	Busy	6	Vent	0	10		
Cryo Off	Off	0	On	4	Off	0	Vent	0	4	A	Foreline off
	Off	0	On	4	Off	0	Busy	15	19		
Off Wait	Off	0	Off	0	Off	0	Busy	15	15		
Go Off										A	Switch off
Go Vacuum										A	Switch vac
Go Vent										A	Switch vent

One of ordinary skill in the art would be able to code the appropriate computer software and construct and connect the actual input/output circuitry for connection to a physical system (where needed or desired) in light of this disclosure. The computer software may be coded in any one of a number of computer languages. Examples of suitable languages are C++ or Pascal (see Figure 7). The graphical user interface may take any one of a variety of forms. For instance, a typical graphical user interface uses different colors where, for instance, each node may have a different color indicating its state. A variety of icons are usable for the graphical user interface. The actual icons and/or graphical user interface symbols are a matter of choice.

Therefore, the present invention is not limited to the embodiments of this disclosure, which is illustrative and not limiting, but includes modifications and additions thereto and is defined by the appended claims.

WHAT IS CLAIMED

I claim:

1. A computer implemented method for an automated physical system, the method comprising the acts of:

providing a plurality of elements, each element being expressed as a computer implemented object associated with a part of the physical system;

arranging the elements in a hierarchy;

defining a plurality of states for each element; and

combinatorially aggregating the states for all of the plurality of elements, thereby to determine a state of the physical system.

2. The method of Claim 1, wherein at least some of the elements are associated with physical devices that are not computer implemented.

3. The method of Claim 1, further comprising the act of aggregating a plurality of lower level of states to determine each of the states.

4. The method of Claim 2, further comprising the act of coupling some of elements to an associated physical device by a computer implemented device driver.

5. The method of Claim 4, further comprising the act of coupling the device driver to the associated physical device by input or output circuitry.

6. The method of Claim 4, further comprising the act of coupling the device driver to an interface database.

7. The method of Claim 1, wherein the act of aggregating is performed in one of a simulation mode or a control mode.

8. The method of Claim 1, wherein the act of aggregating comprises adding or multiplying.

9. The method of Claim 1, wherein the act of aggregating the states is performed on the results of functional transformations of those states.

10. The method of Claim 1, wherein one hierarchy of the elements includes a plurality of nodes and at least one endnode having no subordinate nodes.

11. The method of Claim 1, wherein the states initiate the transmission of commands.

12. The method of Claim 1, further comprising the act of commanding one of the elements to enter a different state.

13. The method of Claim 1, wherein a state is a representation of an analog or digital value.

14. The method of Claim 1, further comprising the act of reporting a structural location of each element.

15. The method of Claim 1, wherein at least one selected element is coupled to other elements to perform a predetermined task of the system.

16. The method of Claim 12, wherein the act of commanding initiates operation of the system so as to change state of the elements.

17. The method of Claim 1, further comprising the acts of:

providing a user interface displaying each of the elements and the state of each element; and

periodically updating the displayed user interface to illustrate operation of the system.

18. The method of Claim 11, wherein the transmission is dependent on at least one condition.

19. The method of Claim 17, wherein selection of an element opens that element to display its content.

20. The method of Claim 17, further comprising the acts of:
providing a library of representations of elements; and
looking up the representations in the library.

21. The method of Claim 20, wherein at least some of the elements in the library are related by positional and orientational transformations.

22. The method of Claim 21, wherein the transformations are applied to transformations in related elements.

23. The method of Claim 17, wherein a representation of an element is dependent on a state of the element.

24. The method of Claim 20, wherein an element to be displayed carries the location of the representation and displays the representation with respect to a window.

25. The method of Claim 1, wherein the state of at least one element is stored.

26. The method of Claim 1, wherein each of the states is a type selected from a group consisting of passive, active, not-active, delayed, timed, wait, and conditional.

27. An apparatus for an automated physical system, comprising:
a plurality of elements, each element expressed as a computer implemented object associated with a part of the physical system;
a hierarchy in which the elements are arranged;
wherein there is a plurality of states defined for each element, and the states are aggregated for all of the plurality of elements, thereby to determine a state of the physical system.

28. The apparatus of Claim 27, wherein at least some of the elements are associated with physical devices that are not computer implemented.

29. The apparatus of Claim 27, wherein a plurality of lower level of states are aggregated to determine each of the states.

30. The apparatus of Claim 28, further comprising an associated physical device coupled to some of the elements by a computer implemented device driver.

31. The apparatus of Claim 30, further comprising input or output circuitry coupled between the device driver and the associated physical device.

32. The apparatus of Claim 30, further comprising an interface database coupled to the device driver.

33. The apparatus of Claim 27, wherein the aggregating is performed in one of a simulation mode or a control mode.

34. The apparatus of Claim 27, wherein the aggregating comprises adding or multiplying.

35. The apparatus of Claim 27, wherein the aggregating the states is performed on the results of functional transformation of those states.

36. The apparatus of Claim 27, wherein one hierarchy of the elements includes a plurality of nodes and at least one endnode having no subordinate nodes.

37. The apparatus of Claim 27, wherein the states initiate the transmission of commands.

38. The apparatus of Claim 27, wherein one of the elements is commanded to enter a different state.

39. The apparatus of Claim 27, wherein a state is a representation of an analog or digital value.

40. The apparatus of Claim 27, wherein a structural location of each element is reported.

41. The apparatus of Claim 27, wherein at least one selected element is coupled to other elements to perform a predetermined task of the system.

42. The apparatus of Claim 38, wherein the commanding initiates operation of the system so as to change state of the elements.

43. The apparatus of Claim 27, further comprising:

a user interface displaying each of the elements and the state of each element, and wherein the displayed user interface is periodically updated to illustrate operation of the system.

44. The apparatus of Claim 37, whereas the transmission is dependent on at least one condition.

45. The apparatus of Claim 43, wherein selection of an element opens that element to display its content.

46. The apparatus of Claim 43, further comprising:

a library of representations of elements; and

means for looking up the elements representations in the library.

47. The apparatus of Claim 46, wherein at least some of the elements in the library are related by positional and orientational transformations.

48. The apparatus of Claim 47, wherein the transformations are applied to transformations in related elements.

49. The apparatus of Claim 43, wherein a representation of an element is dependent on a state of the element.

50. The apparatus of Claim 46, wherein an element to be displayed carries the location of the representation and displays the representation with respect to a window.

51. The apparatus of Claim 27, wherein the state of at least one element is stored.

52. The apparatus of Claim 27, wherein each of the states is a type selected from a group consisting of passive, active, not-active, delayed, timed, wait, and conditional.

53. The apparatus of Claim 27, wherein the apparatus is implemented on a first computer, and further comprising a second computer coupled to the first computer, wherein a user accesses the apparatus by the second computer.

54. The apparatus of Claim 27, wherein the apparatus is implemented on a plurality of computers, each computer controlling a portion of the hierarchy of elements.

55. The apparatus of Claim 27, wherein an element which is numerical is a compiled function.

56. The apparatus of Claim 55, further comprising an interface database coupled to the hierarchy, the interface database being associated with the physical system, wherein the function is an entry in the interface database.

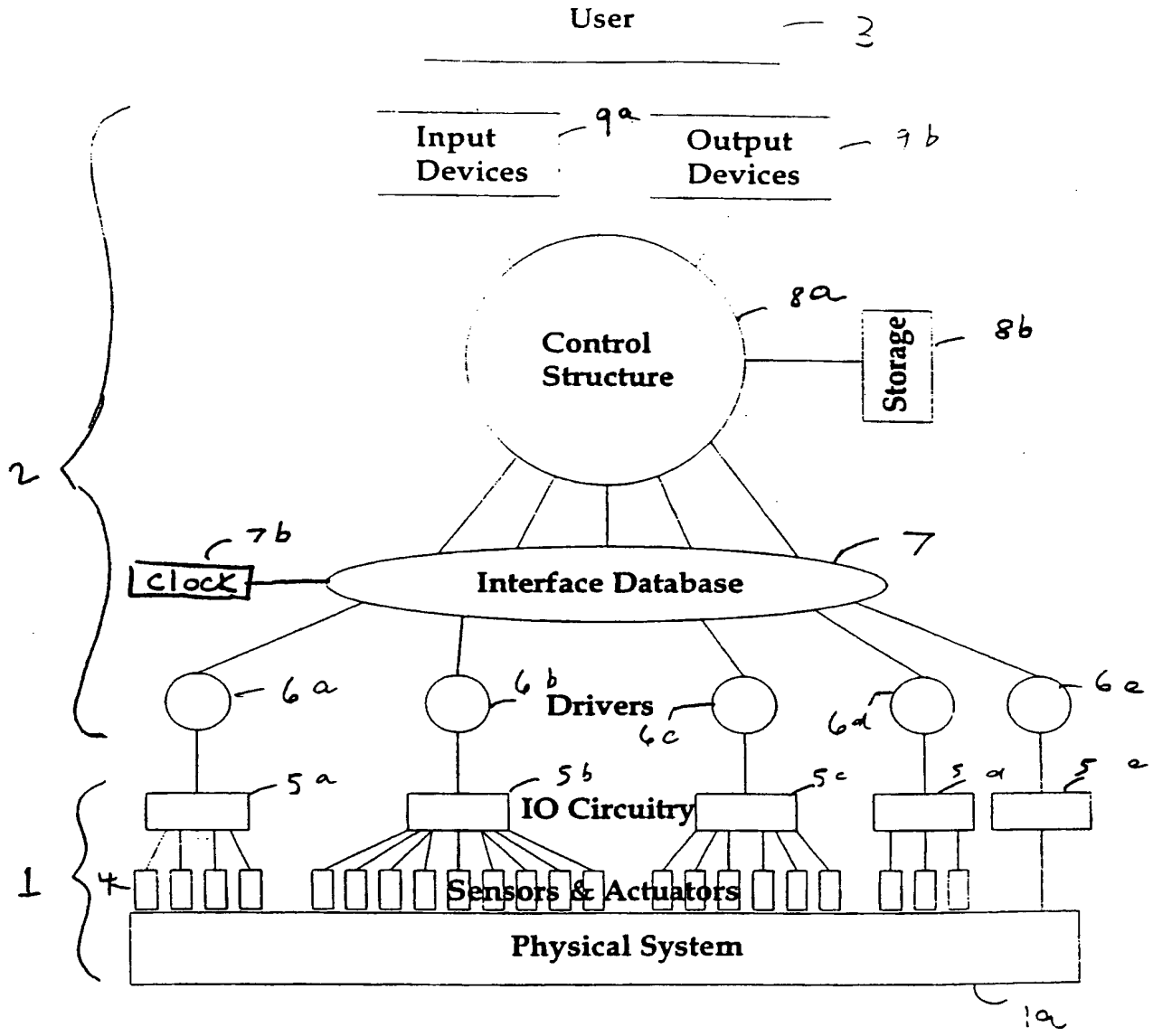


Figure 1

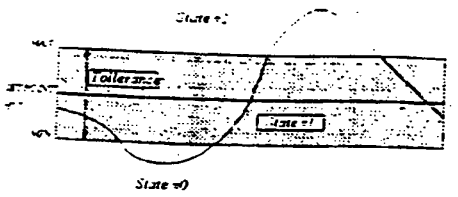


Figure 2a

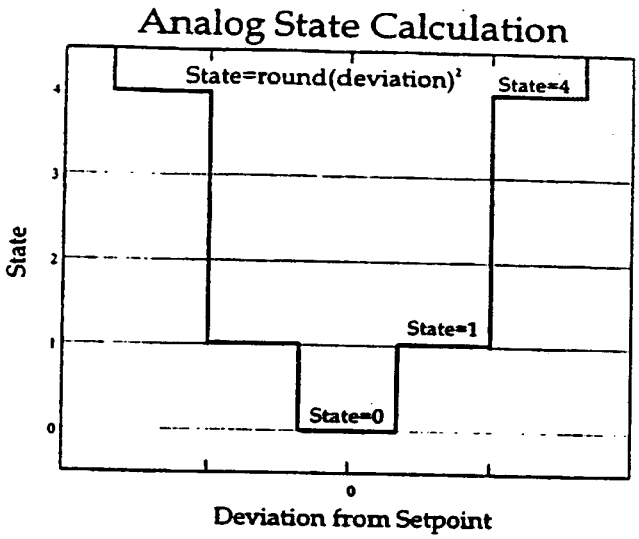


Figure 2b

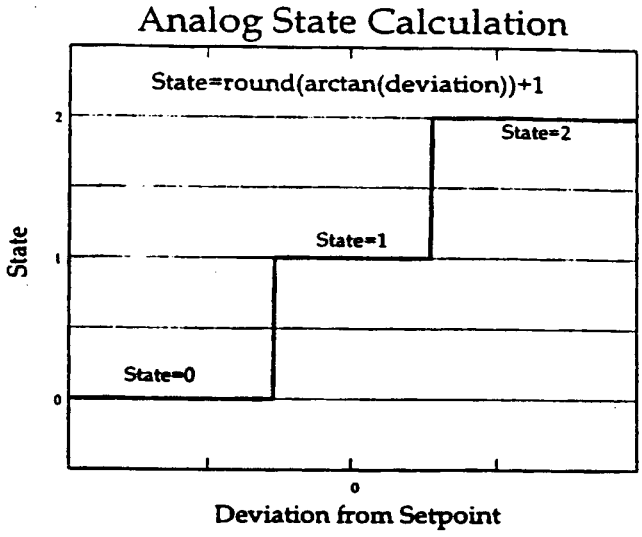

Analog State Calculation
State=round(arctan(deviation))+1
State=2
State=1
State=0
State
2
1
0
0
Deviation from Setpoint

Figure 2c

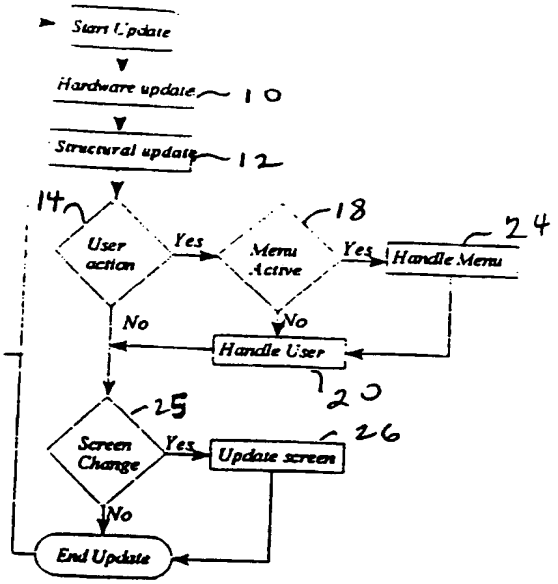


Figure 3

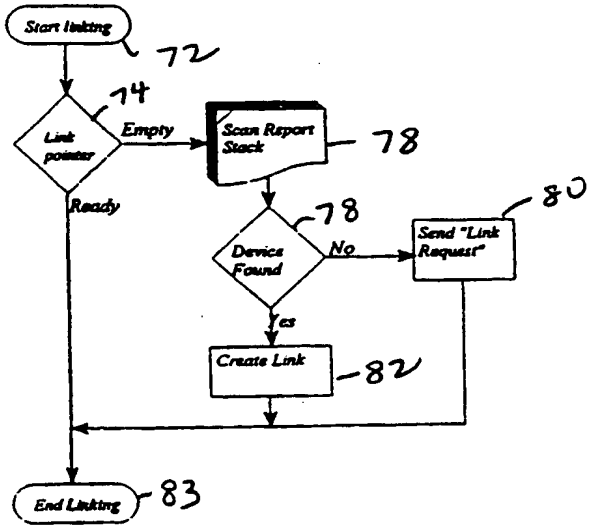


Figure 8

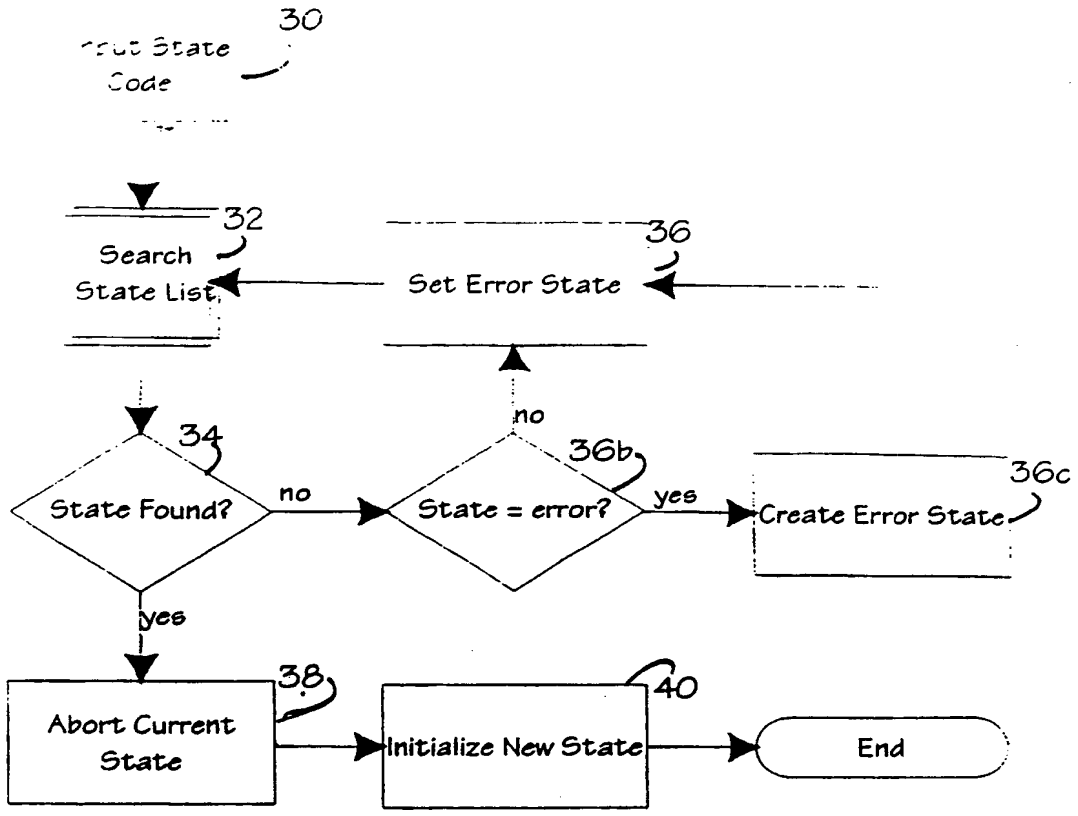


Figure 4

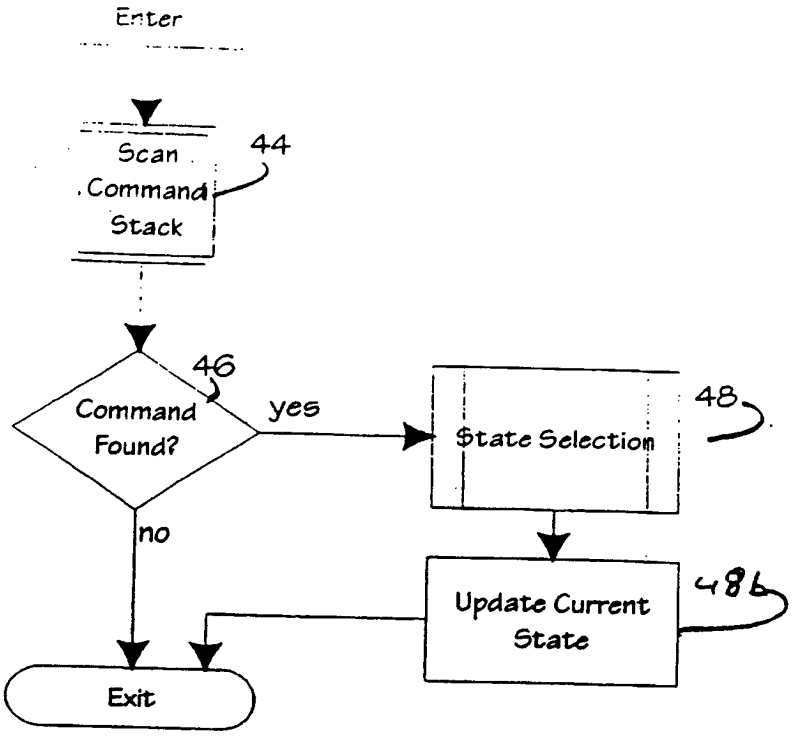

Enter
Scan Command Stack
44
Command Found?
46
yes
State Selection
48
no
Update Current State
48b
Exit

Figure 5a

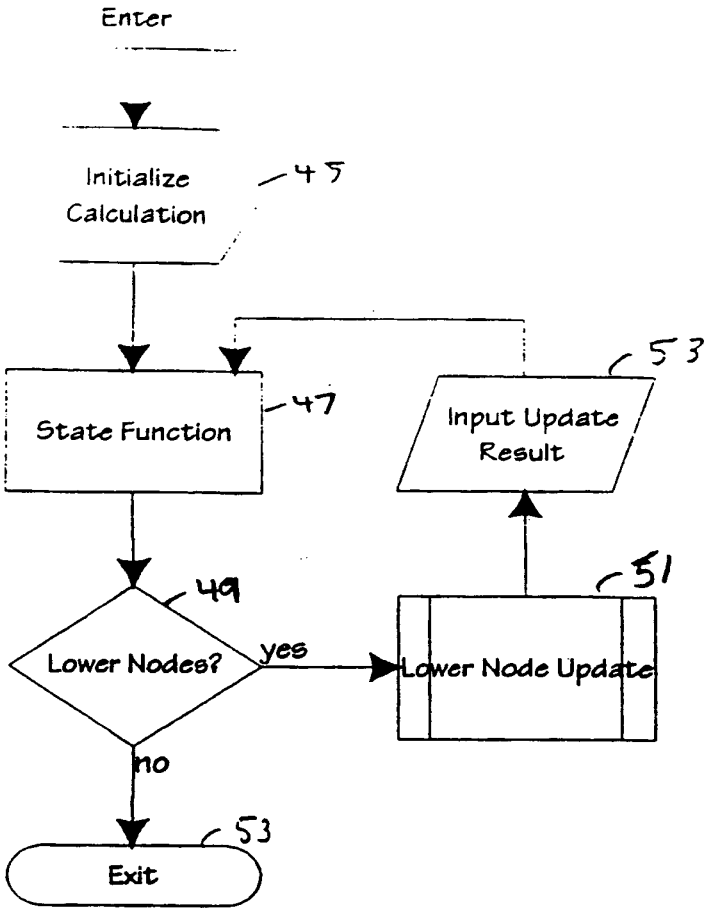

Enter
Initialize Calculation
45
State Function
47
Input Update Result
53
49
Lower Nodes?
yes
Lower Node Update
51
no
53
Exit

Figure 5b

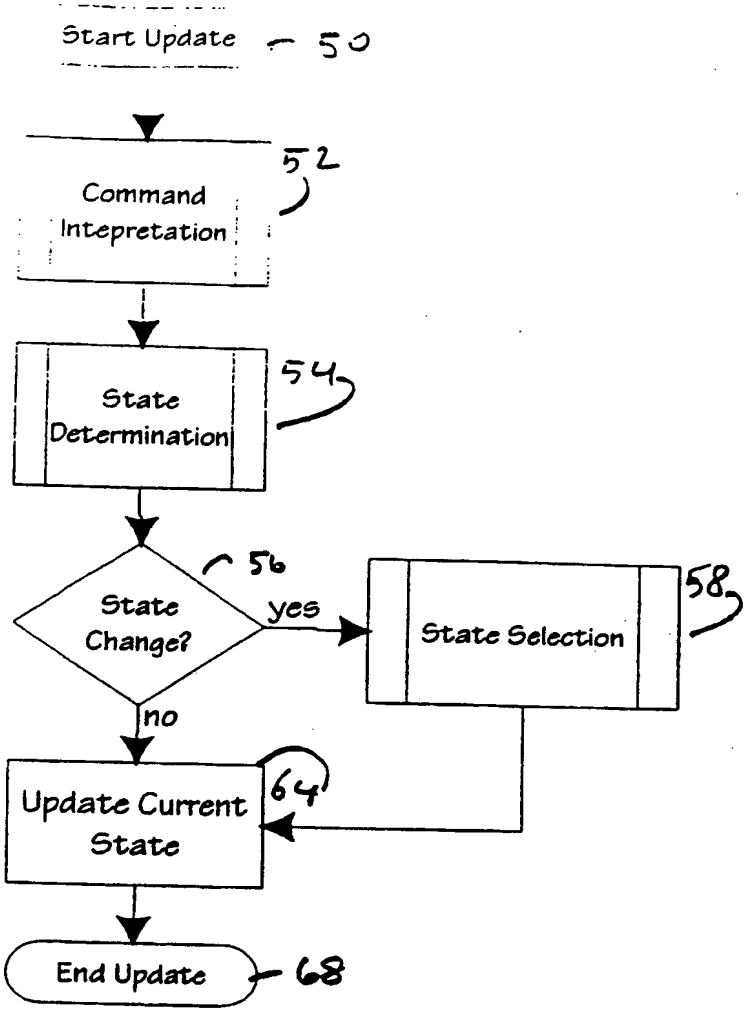

Start Update
50
Command Intepretation
52
State Determination
54
State Change?
56
yes
State Selection
58
no
Update Current State
64
End Update
68

Figure 6

```
FUNCTION Update_Code: Number;
        :
    Initialize;
        :
    IF Command_Found(Command)
        THEN
            Current_State:=Select_State(Command);
            Update_State(Current_State)
        ENDIF;
        :
    New_State:=0;
    FOR i := 1 TO Number_Of_ChildNodes DO
        New_State:=New_State+ChildNode[i].Update_Code;
        :
    IF New_State<>Current_State
        THEN Current_State:=Select_State(New_State);
    Update_State(Current_State);
        :
ENDFUNCTION;
```

Figure 7

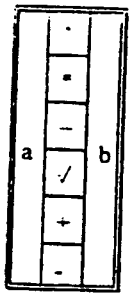


Figure 9a

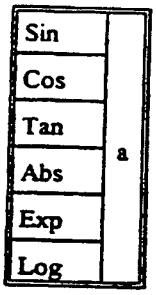


Figure 9b

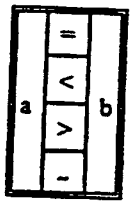


Figure 9c

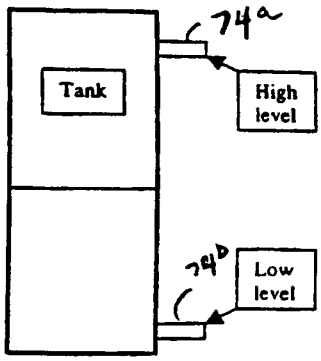


Figure 10

-HF Concentrate Tank
- -Weight Change: *12111*
- -Weight: *12112*
- -Low level: *12113*
- -Overfill: *12115*
- -High Level: *12113*
- -Leak: *12116*

-External Supply: *1213*
- -Supply Pump: *12131*
- -Nitrogen Tank Pressure: *12132*

-HF Acid Mixing System: *1212*
- -Batch Controller: *12122*
  - -Start: *121221*
  - -stop: *121222*
  - -Water Valve: *121223*
  - -Overrun: *121224*
- -Metering Pump: *12123*
  - -Pump Enable: *121231*
  - -Control Pulse: *121232*
  - -Alarm Relay: *121233*

-System HF dil.: *1218*

Figure 12

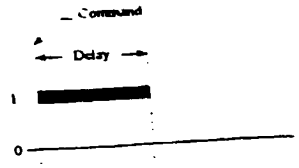


Figure 14

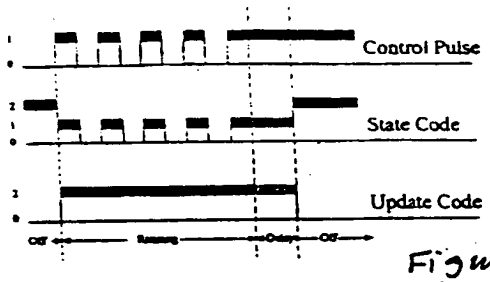


Figure 13

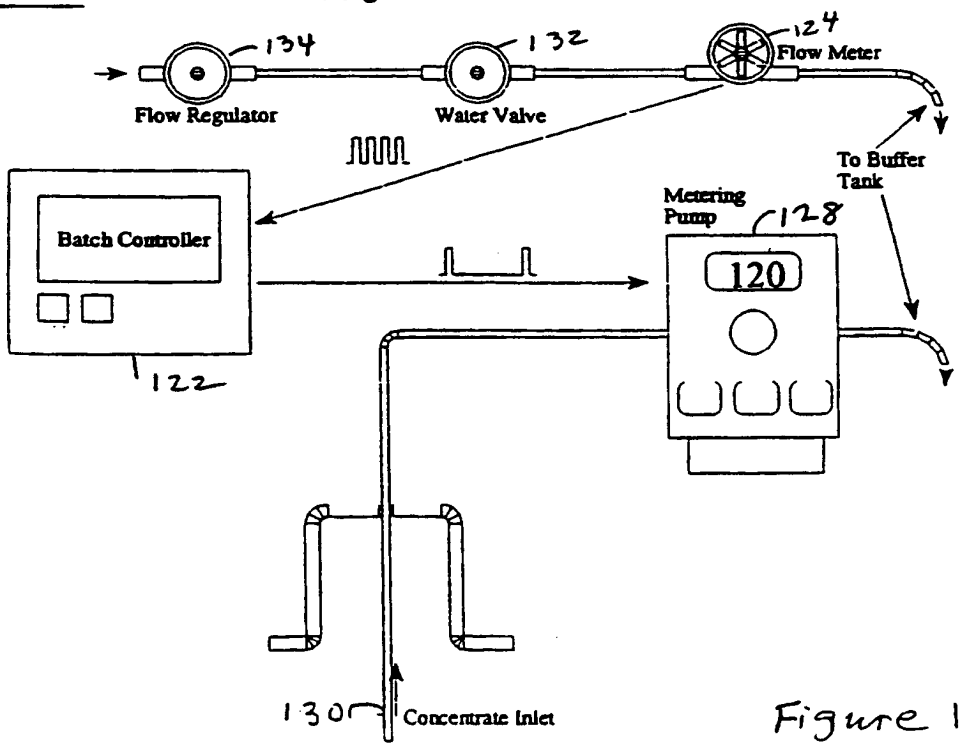


Figure 11

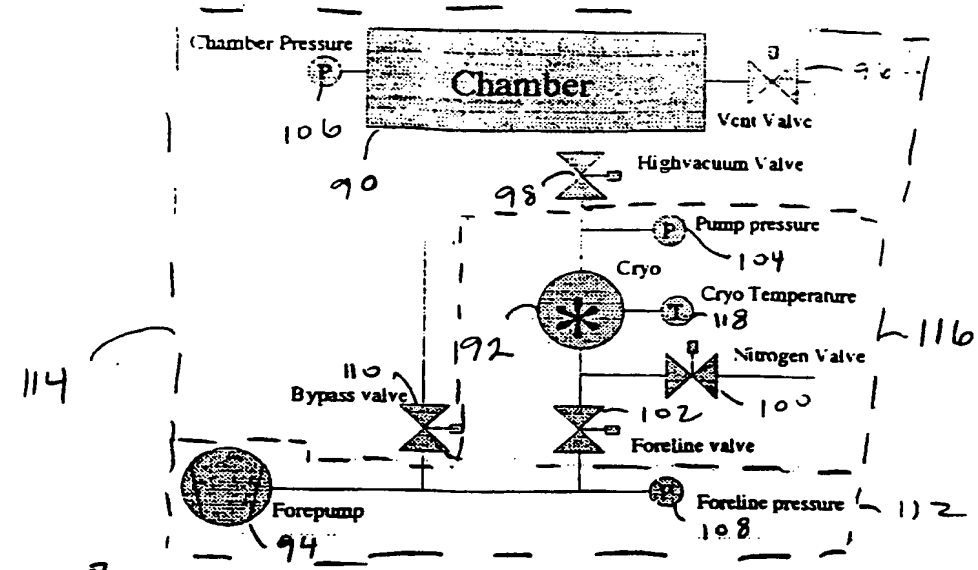


Figure 15

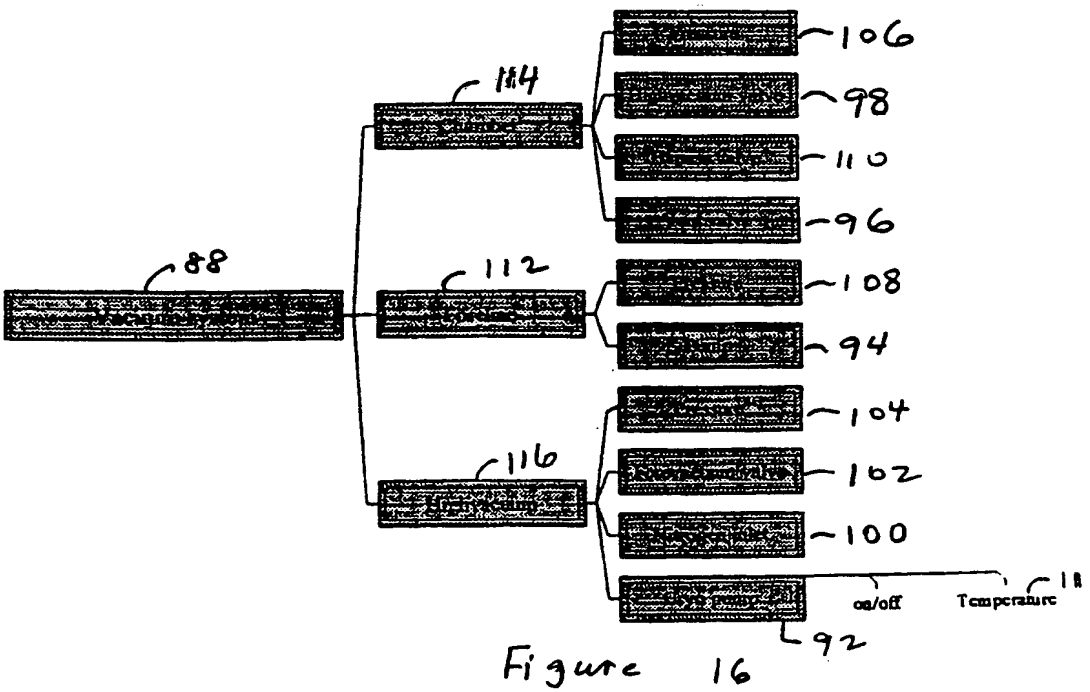


Figure 16

INTERNATIONAL SEARCH REPORT

Internal Application No
PCT/US 00/18179

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G05B17/02

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G05B

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 97 12301 A (FISCHER HORST ;LOEWEN ULRICH (DE); SONST HORST (DE); FREITAG HARTM) 3 April 1997 (1997-04-03) page 4, line 1 -page 7, line 29 ---	1,7,27, 33
X	US 4 965 743 A (BASHAM BRYAN D ET AL) 23 October 1990 (1990-10-23) column 12, line 1 -column 28, line 49 ---	1,27
X	FR 2 724 744 A (ASS POUR LE DEV DE L ENSEIGNEM) 22 March 1996 (1996-03-22) figure 2 ---	1,27
A	EP 0 482 523 A (OSAKA GAS CO LTD ;UNIV VANDERBILT (US)) 29 April 1992 (1992-04-29) ---	
	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
3 October 2000	11/10/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Kelperis, K

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Internal Application No
PCT/US 00/18179

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 812 394 A (LEWIS ROBERT W ET AL) 22 September 1998 (1998-09-22)	

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Internal Application No
PCT/US 00/18179

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9712301 A	03-04-1997	DE 19639424 A	27-03-1997
		EP 0852759 A	15-07-1998
US 4965743 A	23-10-1990	NONE	
FR 2724744 A	22-03-1996	NONE	
EP 0482523 A	29-04-1992	JP 6266727 A	22-09-1994
		US 5420977 A	30-05-1995
US 5812394 A	22-09-1998	NONE	

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 January 2001 (04.01.2001)

PCT

(10) International Publication Number
WO 01/001207 A1

(51) International Patent Classification[7]: G05B 17/02

(21) International Application Number: PCT/US00/18179

(22) International Filing Date: 30 June 2000 (30.06.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/346,107 30 June 1999 (30.06.1999) US

(71) Applicant: ETEC SYSTEMS, INC. [US/US]; 26460 Systems, Inc., Hayward, CA 94545 (US).

(72) Inventor: SCHOLTE VAN MAST, Bart, G.; 5945 Corte Espada, Pleasanton, CA 94566 (US).

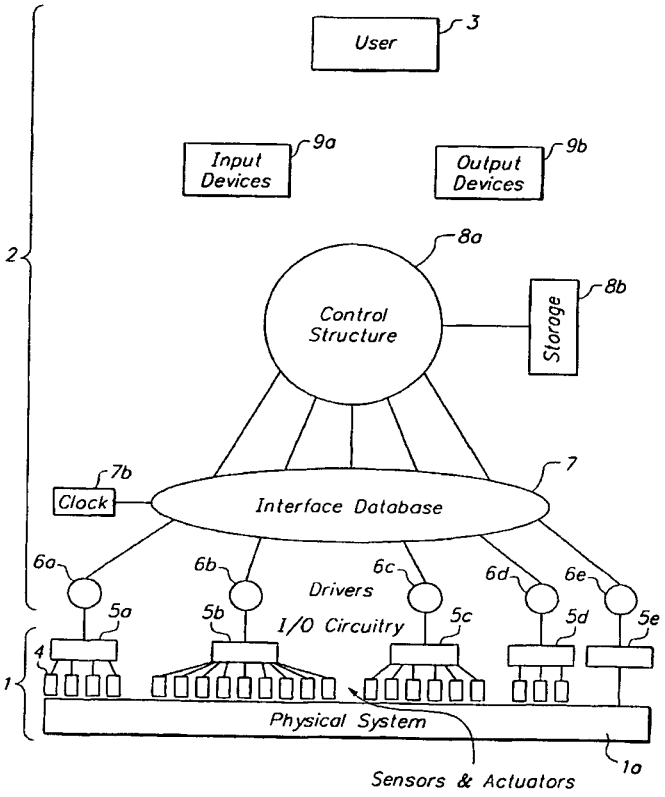
(74) Agents: BERNADICOU, Michael, A. et al.; Blakely, Sokoloff, Taylor & Zafman, LLP, 12400 Wilshire Boulevard, 7th floor, Los Angeles, CA 90025 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR HIERARCHICAL CONTROL OF CONTINUOUSLY OPERATING SYSTEMS



(57) Abstract: For simulation and/or control of automated physical ("real") systems, a hierarchical control system is embodied in computer software executable on a general purpose computer. This is suitable for control and monitoring of complex physical systems without use of elaborate interface circuitry. An ergonomic user interface (7) graphically closely adapts to the particular automated system which is being simulated and/or controlled. The automated system is partitioned into various sub-elements, each of which typically represents a physical part, for instance, pumps or gauges or tanks in a chemical processing system. These elements are arranged in a hierarchical structure of elements. Each element has associated with it the intelligence and logic, in terms of the computer software, needed for its corresponding physical functionality.The function of each element depends on its current state as defined by the computer software and the states can be changed by commands as in the physical system.

WO 01/001207 A1

Published:
— *with international search report*

(48) Date of publication of this corrected version:
25 July 2002

(15) Information about Correction:
see PCT Gazette No. 30/2002 of 25 July 2002, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND APPARATUS FOR HIERARCHICAL CONTROL OF CONTINUOUSLY OPERATING SYSTEMS

FIELD OF THE INVENTION

This invention relates to control and simulation of continuously operating systems using computer software.

BACKGROUND

Continuously operating systems are common; examples are chemical plants, portions of chemical plants, security systems, automated aircraft, semiconductor wafer processing systems, building HVAC systems, and material handling systems. There are many other examples in both manufacturing industries and other fields. Typically such systems include a number of sensors which sense physical changes, for instance temperature, pressure, speed, flow rates, position, and actuators which control such parameters. The sensors and actuators are coupled to a central controller which typically executes some sort of software. The controller may or may not be a general purpose computer. The controller evaluates the sensor data, adjusts the actuators, and typically provides some sort of a display to a human operator indicating a status of the system. The human operator then may or may not intervene in operation of the system; in many cases the system operates automatically under control of the controller. In these cases the controller is controlling functions of the system.

Alternatively, there are simulations of such systems in which there are no physical sensor inputs or controlled outputs but instead virtual input and output data is provided, for purposes of simulation and/or design of a system. Typically, however, such real or simulated systems require customized software, especially in the simulation regime. Often, systems of this type also require specialized computer hardware. They are thus relatively difficult to program. Often, there is no standardized programming interface but instead custom programs (software). Of course, this increases costs and makes such system relatively difficult to design and implement. Therefore, improvement is needed in such systems for both control and simulation purposes, which typically includes design of such continuously operating systems.

SUMMARY

In accordance with this invention, there is disclosed a generalized method and system for the control and/or simulation of automated (e.g., continuously operating) physical systems.

This control and/or simulation is embodied in computer software executed by a host computer, e.g. a standard personal computer, which controls and monitors complex applications (physical systems) without use of specialized (computer) hardware. Advantageously an intuitive and easy to use graphical user interface adapts closely to the particular physical system being monitored and/or simulated.

The computer software includes a number of elements, each being a computer software object representing a part or plurality of parts of the physical system being controlled and/or simulated. The elements are arranged in a hierarchy, each element being represented by one of a plurality of states. A state of the system is an aggregation of the element states.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 shows a system in accordance with the invention.

Figure 2a shows graphically analog state definitions.

Figure 2b shows graphically an analog state with a "dead band."

Figure 2c shows continuous state transitions.

Figure 3 shows a flow chart for the main application process.

Figure 4 shows a flow chart for the state determination procedure.

Figure 5a shows a flow chart for command interpretation.

Figure 5b shows a flow chart for structural state determination.

Figure 6 shows a flow chart for the structural updating and state calculation.

Figure 7 shows code for the controller state machine.

Figure 8 shows a flow chart for cross link creation.

Figure 9a, 9b, 9c show examples of compiler functions.

Figure 10 shows a tank for holding a liquid in a first example.

Figure 11 shows a chemical dilution system in a second example.

Figure 12 shows a hierarchy for the Figure 11 system.

Figure 13 shows a timing diagram for the Figure 11 system.

Figure 14 shows a timing diagram for a switch of Figure 11.

Figure 15 shows a high vacuum system in a third example.

Figure 16 shows a hierarchy for Figure 15.

The use of the same reference symbols in different drawings indicates similar or identical items.

DETAILED DESCRIPTION

The computer implemented method and apparatus disclosed herein are for the control and simulation of automated (typically continuously operating) physical systems. The present method and apparatus are intended to control, monitor, and simulate complicated physical systems without the use of elaborate dedicated interface or controller circuitry. The method and apparatus provide a user interface that adapts closely to the particular physical system.

In accordance with the invention, a physical (real) system is treated as a combination of interacting physical parts (components). Each part has a software entity counterpart called an element. Each element is expressed (coded) as an object which is the actual software source code used to create an element; hence one object can produce multiple elements. For instance, there is one object which is a generalized binary output, e.g., a real valve in a physical system has a corresponding software valve element which is a particularized version of the output object. The elements are combined (conglomerated) into a hierarchical structure. The function of an element depends on the current state of the element. The method is to assign a state to an element from the combined states of a collection of elements. The method allows fast and reliable state determination.

Each element in the system has an associated graphic (graphical user interface) representation. These graphics can be designed or selected to correspond to the real hardware or schematic diagrams of the physical system. The graphic representation follows the hierarchical structure of the physical system, displaying all the available information on each hierarchical level. This allows the user to browse through the hierarchy.

Figure 1 shows graphically the physical portion of the system 1, the software and computer portion 2 which is executed on a host computer, and the human user 3. The physical portion includes the physical system 1a, the sensors and actuator 4, and the I/O (input/output) circuitry 5a, ..., 5e. The software and computer portion 2 includes the device drivers 6a, etc., the interface database 7 and its associated clock (timer) 7b, the control structure 8a which accesses associated storage (e.g., hard disk) 8b, and the input devices (e.g., mouse, keyboard) 9a, and output devices (e.g., computer screen) 9b with which user 3 interacts.

In one embodiment, the software and computer portion 2 is executed on multiple host computers linked into a network rather than a single computer. Even execution of the control structure 8a may be distributed over several networked host computers. The user 3 access may be by a remote computer through a network. Figure 1 will be better understood from the following disclosure.

Control Logic

The software of control structure 8a is organized in a hierarchy of objects of different types. The fundamental object types are used to create the control structure elements where the same coded objects are used multiple times to represent multiple elements. With these elements, the structural functionality of the physical system 1a is described. The following object types are present in the control structure 8a:

Basic Elements

The entire control structure is based on a few basic element types. Each object in the control structure is derived from these basic types.

Element - Each item defined in this portion of this disclosure is referred to as an element. An element is any kind of software object that is defined in the system. Elements have different forms and functions depending on their definition. Elements can be stored and retrieved in storage 8b. Elements have a graphical representation on the output device (screen) 9b.

List - All the elements are arranged in lists of multiple elements. Lists can be displayed, modified, stored and restored. Lists are dynamic structures without predefined length, format, or dimension.

Graphical element - A graphical element holds the coordinates and the form of any graphic representation. These representations are points, lines, boxes, circles, text, icons, etc. Graphical elements are displayed on the screen with respect to a window as described below.

Icon - A list of graphical elements is referred to as an icon.

Icon library - The graphical library is a list of icons. In this list, each icon is defined only once. Since in the list that makes an icon, references to icons within the icon library are allowed, the graphic library is an internally linked, annotated structure.

Window - A set of translation vectors, scale factors and other image transformations. Windows are used to display each icon or element in its screen representation. A window could shift the element on the screen and can scale the element representation to the current environment. The transformation, applied by the window, is also applied to the windows in the transformed icon. This allows the use of icons within icons to infinite levels. An icon within an icon is drawn using repeatedly transformed coordinates.

Screen - A screen contains a list of items that are currently visible on the screen. The screen will update an item on the screen every time its representation changes.

Hierarchical Elements

The elements defined in this subset are the basic structural units and, when combined, form the hierarchical representation of the control structure. Elements combined in a structure are referred to as nodes.

Node - A hierarchical node is an element with one connection to higher levels and multiple connections to lower levels. Downside connections are arranged in a list. The number of levels is not limited. The control structure 8a is made by arranging a number of nodes. Nodes can be defined to perform tasks as function of their present state. In this form, nodes act as subsystems. To identify nodes of any type in the control structure, each node has a unique identification label.

Child node - A child node is a node seen looking down the hierarchy.

Parent node - A parent node is a node seen looking up the hierarchy.

Endnode - An endnode is a hierarchical node with only one connection to higher levels and no connection to lower levels. Endnodes are nodes at the lowest end of a structural branch. Several types of endnodes can be predefined. Each endnode definition gives the endnode certain specific functionality. Endnodes are used to make connections to the hardware of the system as IO (input/output) devices but also have other functions. Endnodes perform tasks in the same way that nodes can but the number of states of an endnode is typically lower.

Cross link - Cross links are direct, one directional links between two nodes. Since a link can be made with any node, it can be used to bypass the limitations of a hierarchy. After a cross link is established, all the information of the node at the end of the links is available to the node at the beginning. Cross links are also used by the function compiler as defined hereafter. Cross links are made as needed. Established links can be maintained after use.

Guest node - When a node is linked to other nodes by a cross link, this node is called the guest node of the element to which it is linked.

Hostnode - The node that carries the link. Specific types of hostnodes can be defined.

Shadownode - A shadownode is a hostnode with no other function than to be a transparent representation of another node or endnode in the structure, its guest node. Shadownodes can be used to link an entire branch of the control structure to another, without redefining this branch. A shadownode of an endnode is useful to make certain information immediately available throughout the structure. Commands, as defined hereafter, sent to a

shadownode are directed towards the guest node. If no cross link is available, commands to this node are ignored.

Dynamic node - A dynamic node is a shadownode without a pre-defined destination of its cross link. If a cross link is established, the node acts as a normal shadownode. Establishing the cross link for a dynamic node is initiated by sending it a predefined command that carries the identification label of the desired guest node. It will take the identification label and start the normal cross linking procedure as defined hereafter with this label as defined for the shadow node. This implicates that if the node has an established cross link, the new link will not be formed. Terminating the cross link is done by predefined command.

Access Node - An access node is a dynamic node that has a connection to an interface device.driver 6a, etc., either via the interface database 7 or directly. All information available to the access node can be transmitted over the communication line by sending a predefined command to this node. Specific parameters can be selected by adding the parameter identification to the command (e.g., ParID = 1: node state). If the cross link is not established, the transmitted data is taken from the access node itself. All information entering the node through the communication line is posted as internal commands. This automatically allows the access node to create and terminate cross links throughout the control structure. The entire control structure can therefore be addressed through the communication line. Communication protocol and error checking are defined in the communication interface device driver.

Active Elements

Hierarchical nodes define the static framework of the control structure, and active elements are used to cause action in the control structure. They make direct status-related communication between hierarchically linked nodes and to make nodes perform certain tasks. They are also used to create a command-action sequence possible of non-hierarchically linked nodes.

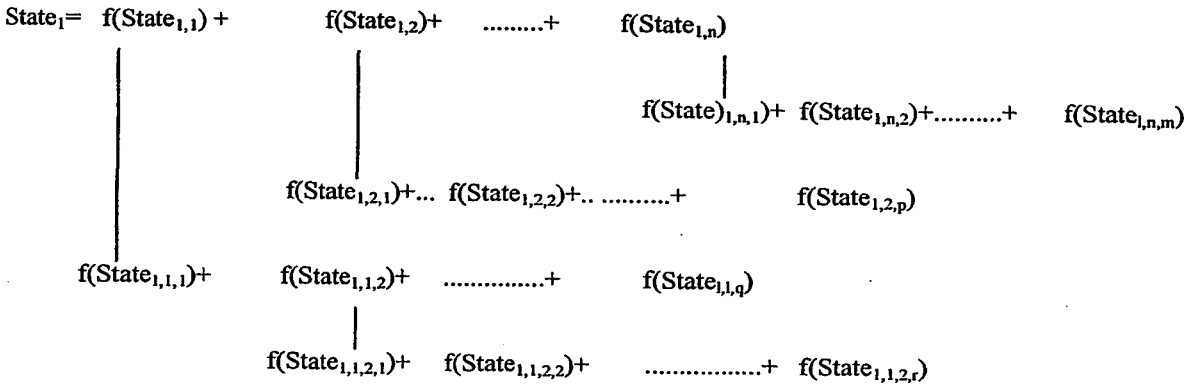
State List - All the active elements are arranged in lists of multiple active elements. Every node carries a state list. These lists are available to the nodes to select relevant information.

States - States are active elements that define the current situation and function of a node. Every state has at least one state code. The state code is a numerical representation of the state and defines in what state the node will be in a particular situation. Each state code should only be addressed once. Multiple states with identical codes lead to unpredictable state selections.

Each state has one update code. The update code is therefore dependent on the state of the code. The update code is used to transmit the present state of this node to the hierarchical level above. With these two parameters the actual situation of the node is determined. For endnodes, the determination function of the state is device-specific. For nodes (or subsystems) the state is in one embodiment calculated by the expression:

$$State = \sum_{k=1}^{m} Update_k(State_k)$$

in which k is the number of the child node and m is the total number of attached nodes. Since $state_k$, the state of childnode "k", is calculated with the same expression, the complete structure is updated by calculating the state of the highest level node, of which there is only one. The actual calculation can be represented as:



in which f is the relation between state and update codes, and n, m, p, q, r are indices. Both the state and update code range in value (in one embodiment) from 0 to 255.

The above definitions are mostly logical AND operations. To make it possible to implement logical OR relationships, each state can have more than one state code. The state is appointed when the state calculation leads to a state code in this list. Another way to obtain an OR relation is to define multiple states with the same update code. The OR relation is then established one level higher in the hierarchy.

In another embodiment, instead of the state aggregation being a sum of the states, it is a product (multiplication) of the states, or a combination of a sum and a product.

To accommodate the human user 3, each state has associated displayed text and has a specific effect on the representation of the nodes. In one embodiment, the node is displayed with the current state's name, and every child node is displayed in a state color of the child node.

State Types

Passive state - If a state is only used to transmit information inside the hierarchy through its update code, this state is passive.

Active state - An active state transmits at least one command, as defined hereafter, directly after it has been selected. This state carries its own list of commands. All the commands in this list are transmitted at once. It continues transmitting until the state is aborted and another state becomes current.

Not-Active state - This state is similar to the active state, but it transmits its commands when it is aborted.

Delayed state - A timed delay postpones the transmission of its command list until a certain time has expired. The timer starts when the state becomes active. When the state is aborted before this time was reached, no actions are taken.

Timed state - A timed state postpones the transmit of its command list until a real time clock of the host computer indicates a certain time and/or date. When the state is aborted before this time was reached, no actions are taken.

Wait state - A wait state utilizes the possibility to create cross links throughout the control structure. It only transmits the command list after the node at the end of its cross link reaches a certain state. If the state is aborted before this happens, no action is taken. The first time the state is active it establishes the necessary connection.

Conditional state - The conditional state also uses a cross link connection to another node. This state has two associated command lists. Which command list is transmitted depends on the state of the node at the end of the cross link. Conditional states act immediately after the successful creation of the cross link.

Commands

Commands force nodes into a state. Commands are transmitted by active states of any type and can be sent to any node in the structure. The destination of a command is declared by entering the identification label, e.g., number of the destination node. The actual command is the code of the state that this node should activate. The node will select this state immediately, without checking its IO or calculated/real-time state. The node performs all the actions linked to this state. Commands usually only have state information but can carry additional parameters when this is needed. These parameters can be any kind of number, e.g., analog set points but can also be structural information.

Command list - A list of commands that is linked to a certain state or element. Command lists are transmitted as one unit, but each item of the command list can be retrieved individually.

Command stack - To enable the commands to be transmitted throughout the structure, all commands that are transmitted are stored on a central command stack. This stack is scanned by every node before it makes its update calculation. If a node finds a command with its identification number, it is put in this state immediately. The command is then removed from the command stack. It is preferred to allow nodes to send commands mainly to their own child nodes. This keeps the stack low.

Reports - Reports are shadow-representations of a node. A report that a node transmits contain the nodes identification code and the structural location of this node. Transmitting a report can be forced by sending a predefined command to a node. Reports are used to form cross links between nodes.

Report stack - To ensure that reports are available throughout the structure, all reports that are initiated are stored on a central report stack. This stack is scanned by every node that is trying to establish a crosslink. If a node finds a report with the desired identification number, the structural information in the report is used for the creation of a cross link. The report is then removed from the report stack.

Control elements

With the elements defined above, it is possible to build an active hierarchical structure. The basic elements of these structures are nodes and child nodes with states. These elements are control elements.

Controllers- Controllers are nodes that contain all the information in their states to perform a certain task. Controllers can be displayed as an icon with a sub-icon for each sub-controller it manages. Controllers are completely user defined and have no intrinsic intelligence.

Systems - A system is a controller like any other, except that it displays its descendants as controllers. The descendants are updated continuously with their full representation. This enables the user to monitor several controllers simultaneously.

Devices

The intended purpose of the control structure is to manage a large number of devices in an organized way. These devices are usually endnodes with all the necessary states. Since devices are the software link to the physical (real) world, they include a number of features

that enables them to perform a certain task. In most case, this means that a few of the, e.g., 256 (0 to 255) states that a device can have are predefined. State transitions are then driven by events in the physical portion 1 through interface device drivers 6a, etc.

Since device state transitions are event driven, it is possible to put a device in a different state than is the actual representation of the interface by a command. After a physical event, the device representation will again be in accordance with the physical interface.

Transitional devices - A transitional device is only capable of making state transitions as a result of commands. These devices are useful to set controllers into certain states, without having to change the actual hardware platform. This enables the user to let the system react differently to identical situations.

IO devices - IO (input/output) devices are devices that represent the value of items in the interface database in a few predefined states. They may address these items individually. The types of IO devices are as follows.

Digital input devices - Digital input devices have two predefined states corresponding to their input signal. Logical low will yield state code "0", logical high will result in state "1". The source of the digital signal is determined with a parameter set "channel" and a "line" parameter. "Channel" defines the slot and "line" appoints the bit on this slot of an item in, for example, the interface database.

Digital output devices - Digital output devices have two predefined command states that involve switching an item in the interface database. This is similar to the digital input.

Analog input devices - Analog input devices represent an analog input voltage (signal). This voltage can be scaled using a scale function. Analog input devices may also contain a setpoint and a maximum relative tolerance. With the actual analog input signal these numbers determine the state of the device. The predefined states are declared as represented graphically (in an example) in Figure 2a or by any transfer function. Figures 2b and c show particular rounding function to determine a state value from an analog deviation value. The signal input source is defined by the "channel" parameter. This number represents the source of the analog signal.

Analog output devices - Analog output devices have several predefined commands, all carrying parameters. These commands are used to set, increase and decrease the analog output value of the device. The state of an analog output device is determined similar to that for the analog input.

Linked devices - Linked devices utilize the cross linking possibly. A linked device processes the data of one or more other nodes.

Arithmetic devices - Arithmetic devices have an extra field where a string representing any calculation can be entered. The function is compiled (see description of function compiler below), and is used in a compiled format during runtime. The result of the calculation is the current value of the device. State determination is performed as for an analog device.

Change devices - Change devices monitor absolute change of the value of a guest node. The device is reset at a certain state and it stores the value of the guest node at that time. The device continuously monitors the difference between the stored and the present result of the guest node. The state determination is the same as for an analog device.

Derivative devices - Derivative devices are normally linked to an analog device. It accepts the actual value of the analog device and differentiates it with respect to time. The number of data samples that the device uses is variable. The actual derivation uses the following linear regression approximation of the curve slope in time:

$$Slope_{(t_n=t_o)} = \frac{n \cdot \Sigma value_n \cdot t_n - \Sigma value_n \cdot \Sigma t_n}{n \cdot \Sigma t_n^2 - (\Sigma t_n)^2}$$

Where n is the number of data points, t is the time since t=0 and "value" is the actual value of the analog device. The device state is defined as for analog devices. See the graph of Figure 2c for an example.

Integrating devices - Integrating devices calculate its present result using the following function:

$$Integral_t = integral_{t-dt} + Value_t \cdot dt$$

The state determination of this device is identical to the state determination of an analog device.

System devices - System devices are devices that have special capabilities with respect to the user or system. These devices are usually displayed as a descendant of the top-level system controller.

Clock devices - Clock devices display the current value of a real time clock. The value is updated whenever a state change occurs.

Speed devices - Speed devices display the number of times it has been updated since the last state change.

Memory devices - Memory devices read the system memory. This device is especially useful during system development. The total memory usage of the system structure can be monitored. It is also possible to see if any unaddressed commands are transmitted. This leads to a continuous decreasing of available memory. This device is updated when a state change occurs.

Message - Message devices keep track of all the messages that are sent from the hierarchical structure during runtime. The state descriptions of this controller are the actual message text. It is still possible to connect an action to a certain message. The difference is that these commands are only transmitted once. With the message text, the accompanying command name (usually an indication of the source of the message) and the time of receiving the message command is displayed.

All messages are stored in a text file for later backtracking. The message file also contains the date of the message.

Manual commands

Entering of commands occurs in two different ways. Both are linked to a node and are only available in that node. The given commands however, are treated as if it where internal system commands send by states. Thus, manual commands are not limited to the node that carries the command item.

Command menu - This menu is hidden under a representation displayed on the node whenever the list is not empty. The list is opened when the user selects this button. The commands in the list are defined during configuration of the system. The user can select a command from this list. The selected command will be put on the command stack immediately. A node can carry more then one command list.

Command buttons - A command button is displayed on the node as a single object. This button represents a command list. When the user presses the command button, the complete command list is placed on the command stack. The commands in the list are defined in the configuration. A node can carry more then one command button.

Interface Elements

An illustrative example of an interface platform (see Figure 1) used in the present system is a commercially available National Instruments SCXI-based chassis. The actual software link from the host computer executing control structure 8a to the chassis is via the hardware device drivers 6a, etc., (which themselves are software but are hardware dependent). Device drivers 6a, etc., continuously update the software representation of the interface chassis

in the interface database 7. This software representation depends on the configuration of the chassis.

Chassis - The chassis is the collection of IO circuits 5a, etc., (also referred to as "boards" for circuit boards as described hereinafter).

Analog input board - The analog input board device driver manages, e.g., 32 channels of analog input as presented by an analog multiplexer board. The channels are periodically scanned during runtime of the system. Each channel can be sampled several times before the average value is written in the interface database. To avoid cross channel overlap, the driver switches to the next channel after measuring but then leaves the analog measuring procedure. During the next analog update cycle the then selected channel is updated.

Analog output board - Complementary to the analog input board.

Digital input board - The digital input board driver updates all, e.g., 32 bits of the digital input database to agree with the bitmap of the corresponding physical inputs during each updating cycle.

Digital output board - The output board driver updates all, e.g., 16 channels of the digital output. The lines of the board are set as the bits are present in the digital output database for this board.

Serial IO board - This IO circuitry supports serial communication.

Implementation Methods

The basic types and functions of the elements in the software system are described above. The following describes the interaction between these elements, including processes ("procedures") used by and with the elements to perform their tasks.

Main Method - The main runtime cycle of the software system includes several procedures (see Figure 3). The first procedure 10 is the hardware interfacing update. Next the control structure is updated at 12 to reflect the current hardware situation. If there is a user input device action at 14, - e.g., a key or a mouse button is pressed on the host computer - the user input is handled. Depending on whether the menu is active or not at 18 the user is allowed to perform different actions. If the menu is not active, the user can only browse through the structure or enter manual commands into the structure at 20. If the menu is active, at 24 the runtime editor is active and will interpret the user action. In this mode, the parameters can be altered. If the screen representation of visible elements has changed at 25, these representations are updated at 26.

Commands and state transitions

State selection - A state is selected by entering the state selection procedure by inputting a specific state code at 30 (see Figure 4) which illustrates this state determination procedure. This code is a command number or a determined state. In the state selection procedure, the state list that accompanies the current node is searched at 32 for the state with the entered identification code in its code list. If the state code is found at 34 ("yes"), the state is selected. The present state of the device is then canceled by the abort sequence 38 determined in this state. After that, the new state is initialized at 40.

If a state code is entered in the state determination procedure that is not found at 32, the state determination procedure restarts its search at 36b with a default error state code. This state code is then set at 36, after which the state list is searched again. If the error state is not found, the procedure creates this state at 36c. The state is added to the state list of the device as a passive state with a pre-defined update code.

Command Interpretation - The first step that each device takes in its update cycle is the scanning of the command stack for any incoming commands, as shown in Figure 5a illustrating the command interpretation process. The scanning of the command stack at 44 is abandoned whenever a command is found at 46 or at the end of the stack. This means that during one updating cycle only one command is taken. If there is more then one command for one specific device, they are handled in subsequent updating cycles. If a command is found at 46, the command code number is taken as the code to enter the state selection procedure at 48 followed by a current state update at 48b. The state selection (determination) process 48 is shown in detail in Figure 4. The state that is identified by the code is activated immediately.

Structural state determination - The structural state determination function, as defined above, allows rapid and explicit state determinations. To make a state determination, a node asks all its child nodes to report their update codes, e.g., calculate their states. The node takes the total sum of these codes. This summation is the number that the node uses to enter the state selection procedure. The node enters this procedure only when the result of the summation differs from the state code of the current state. This process is illustrated in Figure 5b. The state calculation is initialized at 45 to enter a state calculation at 47. Child nodes are checked at 49 and updated at 51 to report the update result at 53 to 47. Only when all child nodes are exhausted at 49 is this process exited at 53. Subroutine 51 is the identical process for the childnode. This process descends into the hierarchy until the end of the branch is reached, i.e. an endnode is encountered.

Structural updating - The procedures described above are all that is needed for the structural updating and state calculation as shown in Figure 6 beginning at 50. First, each node starts with the command scanning and interpretation at 52 (shown in detail in Figure 5c). Then the actual node state determination is executed at 54 (shown in detail in Figure 5b). State selections 58 (shown in detail in Figure 4) are only performed when needed as determined at 56. This means that the device will only scans the state list whenever the lower levels change, or when a command is entered. In each cycle, the state update is performed at 64. This is done to allow states that have running timers to update their timers or conditional states to check conditions. An example of code (software) for structural update of a node (the controller state machine) is shown in Figure 7 in the Pascal language.

Cross link creation - A cross link creation process (see Figure 8) is initiated at 72 by the element that needs the link. If the link pointer is empty at 74, the element scans the report stack at 78 for the representation of the device it wants to link with. If the element does not find the information it needs, it transmits a pre-defined command at 80 that forces the potential guest node to put its structural information on the report stack. The element then leaves the current linking cycle at 83.

The potential guest node now finds a link request command on the command stack and transmits its coordinates to the report stack. This is the only action that the guest node takes. During the next cycle, the element that needs the link finds the information of its guest node on the report stack. This information is removed from the report stack and stored in the link pointer at 82. After this cross-link creation, all information of the guest node is available to the linking element.

Interfacing - The actual control system is indifferent with respect to the hardware platform (chassis) it controls. To accomplish this, a hardware device driver 6a, etc. (Figure 1) (software) is in the main runtime loop. This device driver updates the interface database 7 of the hardware situation each time it is addressed. It also sends out any changes to the hardware settings made by the control structure 8a. Immediately after this cycle, the physical portion 1 is exactly represented in the database 7. The corresponding item in the interface database 7 is updated in real time. The control structure 8a only uses the software representation by direct reading and writing on the elements of the database 7.

Function compiler - Any number in the control structure 8a can be replaced by a function. In addition to the standard arithmetic functionality, the function compiler can interpret a number of other operations. These operations or functional relations are used to

speed up evaluations. The functional dependence of state parameters may be established by the function compiler. The function compiler may directly influence a state of a node, e.g., the compiled function may be a component in a child node list of a node.

Standard functionality - Examples of standard arithmetic operators in the compiler are shown in Figure 9a, where 'a' and 'b' are numbers or other functions. Examples of standard mathematical functions in the compiler are shown in Figure 9b.

Boolean functionality - Boolean operators in the compiler are shown in Figure 9c. The result of these evaluations is a real number with value '0' if false and '1' if true. The result of the '~' operator depends on the setting of a range variable. This evaluation is used to determine whether two real values (a and b) do not deviate more then a certain amount. The actual evaluation algorithm is:

$$\left|\frac{a}{b}-1\right|<\Delta$$

in which Δ is the range setting.

Relative functionality. There are two ways of relative addressing of numbers. The first has the form:

[*DevID*,*Par*ID]

This function returns the current value of the parameter of the device for the identification numbers entered between the brackets. The function creates the cross link when it is updated and the link has not been established yet.

The other form is used to directly address the interface database 7. The actual form depends partly on the format of the elements in database 7, but the general form is:

IOP (ChassisID, SlotID, LineID)

The result of this function is the current input or output value of the addressed item in the database 7. Both methods can be reversed to write new values in the addressed locations.

Editing

The runtime editor is a special re-entering editor whose functions can be inserted in the main cycle. The editing functions change parameters and settings without interrupting the control sequence. The editor also allows the user to move items over the screen to any desired position. It is possible to instruct the system to open more then one viewport to the control structure. This allows the user to have multiple controllers on one screen. All items on the screen can be edited by the editor. Whenever the system is halted, the identification codes of the items on the screen are stored in the default screen item list.

No structural changes are allowed during runtime. This means that no subnodes can be removed or added, and that no node identifications can be changed. However, it is possible to alter the state and command tables of the nodes. The runtime editor is only active when the menu structure is active.

In addition to the functionality of the runtime editor, the structural editor incorporates the structural modification functions. With this editor the system structure is defined or modified. It allows the creation, deletion and modification of structural branches and nodes.

Operational Modes

There are three operational modes:

Normal - The normal mode runs the system as quickly as possible. This mode should be used after the system performs as intended.

Step - The stepping mode allows the user to monitor the system evaluation while it takes place. The updating sequence of each node in the structure is halted until a key is pressed. Then the next node is updated and will be displayed on the screen with all current data.

Learning - In learning mode the system runs normally, until it encounters an undefined state. In normal mode, the default error state is selected whenever this happens. In learn mode the system interrupts its updating sequence. The node that has entered the error state is displayed. A small submenu asks the user if this state should be defined, assigned to an other state or ignored.

Defining the current state enters the normal state definition sequence. Selecting an already defined state will add the undefined state code to the state code list of the defined states. Selecting ignore will no longer cause the system to halt at this node for this state. After completion of the state definition, the system continues running with the new state.

The system does not store the modifications; this is done manually. Clearing the learn mode by selecting another running mode, also clears all the ignore flags.

First Example: Liquid Tank

The basic method described above is elucidated by a simple example. In this example there is a liquid container (tank 72), in which two sensors 74a, 74b monitor the liquid level. One low-level sensor 74b is at the bottom of the tank 72 and a high level sensor 74a is at the top (see Figure 10). Several states have to be defined for tank 72. One wants to be able to detect if the tank is "full", "empty" or somewhere in between. This last state is defined as "ready".

Both sensors 74a, 74b can have two states, "wet" or "dry". These two states are directly related to the digital input ("high" or "low") from the sensor. With these states one defines the following state table for the tank:

Tank	Low Level		High Level		State
	State	Update	State	Update	
Empty	Dry	0	Dry	0	0
Ready	Wet	1	Dry	0	1
Full	Wet	1	Wet	2	3
Error	-	-	-	-	255

The table starts with the assignment of the "Empty" state. Both sensors are "dry". Update code "0" is assigned t both states. If the tank is slowly filled, the lowest sensor switches to "wet". This state is assigned an update code "1" leading to a unique state "Ready" in the state calculation. If the tank is filled further, the high level sensor switches to "wet". Update code "2" is assigned to this state. In summation, this leads to a "Full" state with code "3".

It is possible to assign updated code "1" to the high level sensor's "wet" state, as was done for the low level sensor. However, with the assignment of update "2", an undefined state is inserted for the tank, namely "2". If in the current table, state code "2" is encountered, this would mean that the low level sensor is "dry" and the high level sensor is "wet". Since this is physically impossible, this indicates a defective sensor. If state update code "1" would have been assigned, this discrimination between sensors would not be possible.

This is a simple application of the method which shows that, by properly defining the update functions, unique states are identifiable. Leaving states undefined can serve as an error detection mechanism.

Second Example: Chemical Dilution System

A second and more complex example is a chemical dilution system. The physical system controls the dilution of concentrated chemical with water to the desired concentration and as shown in Figure 11 has two major parts. The first is the batch controller 122 which is, e.g., a programmable pulse divider. The controller 122 (a physical part) receives high frequency digital pulses from the flow meter 124. It divides the number of pulses to a level that is send to the metering pump. Both the batch controller 122 and the metering pump 128

are monitored for proper action. The physical system also has a chemical storage tank connected to inlet 130 from which it takes its chemicals. This tank, and all the sensors and systems around it, including regulator 134 and water valve 132, are controlled.

The corresponding software (control structure) hierarchy is depicted in Figure 12 for the Figure 11 physical system. All the elements are accompanied by their corresponding device identification codes.

The following shows the logical tables that control the diluting system of Figure 11. Tables for the tank and for an external supply unit to fill the tank are omitted for brevity.

a. Metering Pump, Control Pulse. The steering pulse from the batch controller 122 to the metering pump 128 is monitored on a digital input line. Because it is only necessary to know if the pump 128 is running, the actual digital state of the pulse is not important. Therefore, this pulse is converted to a two-state update code that identifies the running state of the pump. This is done by adding a third state to the controller. This state is always made active if any of the other two is active for more then a certain time. These other two states are the default digital input states. Both digital states have the same update, so they are indistinguishable for the higher level controller. The timing diagram for controller 122 is in Figure 13. The corresponding state transition table is:

State Name	Input		Action		State	Update
Low Stroke	Low	0	t	go OFF	0	2
High Stroke	High	1	t	go OFF	1	2
OFF		0/1			2	0

b. Metering Pump. The default state of the metering pump 128 is "OFF". (See the following table.) In this state the pump 128 is disabled and does not start pumping if control pulses are available. Before starting, the pump 128 is set in the "READY" state by closing the "pump enable" relay. If control pulses are available, the pump controller goes "RUNNING". If the internal alarm relay of the pump 128 is released, the pump is set to "OFF-error". If this happens during ready or running states, the pump 128 is disabled.

State Name	Pump enable		control pulse		alarm relay		Action	State	Update
OFF	OFF	0	OFF	0	OFF	0		0	0
READY	ON	1	OFF	0	OFF	0		1	1
RUNNING	ON	1	STROKE	2	OFF	0		3	4
OFF-error	OFF	0	OFF	0	ON	4		4	10
ERROR							MP dis	255	

c. Batch controller Start / Stop relays. The batch controller 122 is controlled by two remote switches. These switches start or stop the controller 122 when pressed for a short time. The corresponding digital output devices are set to fall back to their default position, some time after activation. An external command is the only way that these devices can be activated. The corresponding timing diagram is given in Figure 14. The delay time for the two switches is set to different length. The starting pulse is very short. The stopping pulse is quite long. The state transition table is:

State Name	Input		Action		State	Update
OFF	Low	0			0	0
ON	High	1	t	go OFF	1	start=1 stop=4

d. Batch Controller. The batch controller 122 has remote switches, a steering relay for the DI-water valve 132 and an overrun alarm relay. It only handles the "Running" state as active. The total time in this state is limited. The overrun relay sends out commands independently. It shuts down the system and fires the major alarm when activated. The state transitions are:

State Name	Start Relays		Stop Relays		Valve Relays		Overrun		Action		State	Update
STANDBY	OFF	0	OFF	0	OFF	0	OFF	0			0	0
starting	ON	1	OFF	0	OFF	0	OFF	0			1	0
started	ON	1	OFF	0	ON	2	OFF	0			3	2
Running	OFF	0	OFF	0	ON	2	OFF	0	t	go STOP	2	2
stopping	OFF	0	ON	4	ON	2	OFF	0			6	2
stopped	OFF	0	ON	4	OFF	0	OFF	0			4	0
Command States												
START	Start relays ON										100	
STOP	Stop relays ON										101	

e. Acid Mix System. This controller manages the batch controller 122 and its metering pump 128 as shown in the following table. It is a combination of the batch controller and its metering pump:

State Name	Metering Pump		Batch Controller		Action		State	Update
OFF	OFF	0	Standby	0			0	0
STANDBY	Ready	1	Stopped Standby Starting	0			1	0
starting	Ready	1	Started Running stopping	2	t	STOP	3	2
Running	Pumping	0	Started Running Stopping	2	t	STOP	2	2
stopping	Pumping	0	Stopped Standby Starting	0	t	Alarm 2	6	2
ERROR					t	Stop Alarm 1	255	0
Command States								
STANDBY	Metering Pump Enable						99	
START	Batch controller start						100	
STOP	Batch Controller Stop						101	
OFF	Metering Pump Disable						102	

f. Chemical Blending System. This controller (not shown) lets the complete blending system, including its tank, line and supply, act as one unit. The default state of the controller is Standby as shown in the following table. In this state the system is on line and the mixing system is off. This means that the metering pump 128 is disabled. After a start request, the metering pump 128 is enabled which sets the mixing system standby. The system is now Idle. This idle state has its own timer that puts the mixing system running by starting the batch controller 122.

State Name	Concentrate Tank		External Supply		Acid Mixing System		System Line		action		St	U
OFF empty	empty	0	off	0	off	0	off	0			0	0
OFF Loaded	low, ready, full	1	off	0	off	0	off	0			1	0
Standby	l, r, f	1	off	0	Off	0	on	7			8	0
Idle	l, r, f	1	off	0	Standby	2	on	7	t	go run	10	57
Running	l,r,f	1	off	0	running	4	on	7			12	63
Filling init	l,r,f	1	Press.	57	off	0	off	0			58	0
Filling start	l,r,f	1	filling	63	off	0	off	0	t	stop fill warning	64	0
Filling	l,r,f-inc	69	filling	63	off	0	off	0	t	stop fill warning	132	0
Filling done	l,r,f-inc	69	off	0	off	0	off	0			69	0
ERROR									t	off Warning	255	
start									a	AMS Stb		
stop									a	AMS off		
ON									a	Line On	251	
OFF									a	AMS Off Fill Off Line Off	252	
Tank Fill									a	AMS Off Line Off Fill Start	253	

Third Example: High Vacuum System

A third example is a high vacuum system, common in high vacuum technology. Although much of the functionality of the physical system is normally in the actual hardware, in this example the control system controls each part. The (physical) high vacuum system of Figure 15 consists of a vacuum chamber 90 that is pumped with a high vacuum cryo pump 92 connected to a forepump 94. A number of valves 96, 98, 100, 102, 110 are used for the

operation of this physical system and pressure measurements 104, 106, 108 are installed to allow monitoring. The physical system is partitioned along the dotted lines in Figure 15.

The dotted lines in the physical system overview of Figure 15 are boundaries of hierarchically separated portions of the corresponding software. The corresponding software (control structure) hierarchy is depicted in Figure 16. The function of each element can be displayed in logical tables. The following tables contain the elements' states as well as the sub-elements states and update codes:

Cryo	Temperature		Line		State	Update
	State	Code	State	Code		
Off	Off	0	Off	0	0	0
Cooling	Off	0	On	1	1	
	Busy	2	On	1	3	4
OK	OK	4	On	1	5	12
Off-busy	OK	4	Off	0	4	
	Busy	2	Off	0	2	41

HiVac	Cryo		Foreline Valve		Cryo Pressure		Nitrogen Inlet		State	Action	
	State	#	State	#	State	#	State	#		@	Description
Off	Off	0	Close	0	HiPress	0	Close	0	0		
Evac	Off	0	Open	1	HiPress	0	Close	0	1	T	ERROR
Evac-done	Off	0	Open	1	LoVac	2	Close	0	3	A	Close V_{fl} Cryo on
Cooling	Busy	4	Close	0	LoVac	2	Close	0	6	T	ERROR
	Busy	4	Close	0	HiVac	7	Close	0	11		
Run	Ok	12	Close	0	HiVac	7	Close	0	19		
Run-error	Ok	12	Close	0	LoVac	2	Close	0	14	A	Cryo off Open V_{fl}
	Ok	12	Close	0	HiPress	0	Close	0	12		
Regen 1	Busy	41	Close	0	HiVac	7	Open	20	68		
	Busy	41	Close	0	LoVac	2	Open	20	63		
Regen 2	Busy	41	Close	0	HiPress	0	Open	20	61	A	Close N_2 Open V_{fl}
Regen 3	Busy	41	Open	1	HiPress	0	Close	0	42		
Regen 4	Busy	41	Open	1	LoVac	2	Close	0	44	T	Open N_2 Close V_{fl}
	Busy	41	Open	1	HiVac	7	Close	0	49		
Go on										A	Open foreline
Go off										A	Cryo off Open N_2

Foreline	Pressure		Pump		State	Update
	State	Code	State	Code		
Off	HiPress	0	Off	0	0	0
Pump start	HiPress	0	On	1	1	
	Vac 1	2	On	1	3	2
OK	Evac	4	On	1	5	4
Off-busy	Vac 1	2	Off	0	2	
	Evac	4	Off	0	4	2

Chamber	Pressure		Bypass valve		Highvac valve		Vent valve		State	Action	
	State	#	State	#	State	#	State	#		@	Description
Vent	Atm	0	Close	0	Close	0	Close	0	0		
Pump Forevac	atm	0	Open	1	Close	0	Close	0	1		
	Busy	2	Open	1	Close	0	Close	0	3		
Forevac done	Vacuum	4	Open	1	Close	0	Close	0	5	T	Close Bypass
Vacuum	Vacuum	4	Close	0	Open	6	Close	0	10		
Vac_error	Busy	2	Close	0	Open	6	Close	0	8	A	Close highvac
	atm	0	Close	0	Open	6	Close	0	6		
Go Forevac	Busy	2	Close	0	Close	0	Close	0	2	A	Open Bypass
Forevac	Vacuum	4	Close	0	Close	0	Close	0	4		
Vent busy	Busy	2	Close	0	Close	0	Open	12	14		
	Vacuum	4	Close	0	Close	0	Open	12	16		
Vent done	atm	0	Close	0	Close	0	Open	12	12	T	Close vent
Go On										A	Open Bypass Close Highvac Close Vent
Go vacuum										A	Open Highvac
Go vent										A	Close Highvac Close Bypass Open Vent

Vacuum System	Switch		Foreline		Highvac		Chamber		State	Action	
	State	#	State	#	State	#	State	#		@	Description
Off	Off	0	Off	0	Off	0	Vent	0	0		
Vacuum init	Vacuum	1	Off	0	Off	0	Vent	0	1	A	Foreline on
Pump busy	Vacuum	1	Busy	2	Off	0	Vent	0	3		
Pump done	Vacuum	1	On	4	Off	0	Vent	0	5	A	Highvac on
Cryo Evacuate	Vacuum	1	On	4	Busy	6	Vent	0	11		
Wait Cryo	Vacuum	1	On	4	Cooling	12	Vent	0	17	A	Chamber on
Vacuum busy	Vacuum	1	On	4	Cooling	12	Busy	18	35		
	Vacuum	1	On	4	Cooling	12	Forevac	36	53		
	Vacuum	1	On	4	OK	54	Busy	15	74		
Evacuate Init	Vacuum	1	On	4	OK	54	Vent	0	59	A	Chamber on
Vacuum done	Vacuum	1	On	4	OK	54	Forevac	36	95	A	Chamber vacuum
Vacuum	Vacuum	1	On	4	OK	54	Vacuum	96	155		
Vent Init	Vent	156	On	4	OK	54	Vacuum	96	310	A	Chamber vent
Vent busy	Vent	156	On	4	OK	54	busy	15	229		
Vent	Vent	156	On	4	OK	54	Vent	0	214		
Off Init	Off	0	On	4	OK	54	Vent	0	58	A	Cryo Off Chamber Vent
	Off	0	On	4	OK	54	Busy	15	73		
	Off	0	On	4	OK	54	Vacuum	96	154		
Off Cryo	Off	0	On	4	Busy	6	Busy	15	25		
	Off	0	On	4	Busy	6	Vent	0	10		
Cryo Off	Off	0	On	4	Off	0	Vent	0	4	A	Foreline off
	Off	0	On	4	Off	0	Busy	15	19		
Off Wait	Off	0	Off	0	Off	0	Busy	15	15		
Go Off										A	Switch off
Go Vacuum										A	Switch vac
Go Vent										A	Switch vent

One of ordinary skill in the art would be able to code the appropriate computer software and construct and connect the actual input/output circuitry for connection to a physical system (where needed or desired) in light of this disclosure. The computer software may be coded in any one of a number of computer languages. Examples of suitable languages are C++ or Pascal (see Figure 7). The graphical user interface may take any one of a variety of forms. For instance, a typical graphical user interface uses different colors where, for instance, each node may have a different color indicating its state. A variety of icons are usable for the graphical user interface. The actual icons and/or graphical user interface symbols are a matter of choice.

Therefore, the present invention is not limited to the embodiments of this disclosure, which is illustrative and not limiting, but includes modifications and additions thereto and is defined by the appended claims.

WHAT IS CLAIMED

I claim:

1. A computer implemented method for an automated physical system, the method comprising the acts of:

providing a plurality of elements, each element being expressed as a computer implemented object associated with a part of the physical system;

arranging the elements in a hierarchy;

defining a plurality of states for each element; and

combinatorially aggregating the states for all of the plurality of elements, thereby to determine a state of the physical system.

2. The method of Claim 1, wherein at least some of the elements are associated with physical devices that are not computer implemented.

3. The method of Claim 1, further comprising the act of aggregating a plurality of lower level of states to determine each of the states.

4. The method of Claim 2, further comprising the act of coupling some of elements to an associated physical device by a computer implemented device driver.

5. The method of Claim 4, further comprising the act of coupling the device driver to the associated physical device by input or output circuitry.

6. The method of Claim 4, further comprising the act of coupling the device driver to an interface database.

7. The method of Claim 1, wherein the act of aggregating is performed in one of a simulation mode or a control mode.

8. The method of Claim 1, wherein the act of aggregating comprises adding or multiplying.

9. The method of Claim 1, wherein the act of aggregating the states is performed on the results of functional transformations of those states.

10. The method of Claim 1, wherein one hierarchy of the elements includes a plurality of nodes and at least one endnode having no subordinate nodes.

11. The method of Claim 1, wherein the states initiate the transmission of commands.

12. The method of Claim 1, further comprising the act of commanding one of the elements to enter a different state.

13. The method of Claim 1, wherein a state is a representation of an analog or digital value.

14. The method of Claim 1, further comprising the act of reporting a structural location of each element.

15. The method of Claim 1, wherein at least one selected element is coupled to other elements to perform a predetermined task of the system.

16. The method of Claim 12, wherein the act of commanding initiates operation of the system so as to change state of the elements.

17. The method of Claim 1, further comprising the acts of:

providing a user interface displaying each of the elements and the state of each element; and

periodically updating the displayed user interface to illustrate operation of the system.

18. The method of Claim 11, wherein the transmission is dependent on at least one condition.

19. The method of Claim 17, wherein selection of an element opens that element to display its content.

20. The method of Claim 17, further comprising the acts of:
providing a library of representations of elements; and
looking up the representations in the library.

21. The method of Claim 20, wherein at least some of the elements in the library are related by positional and orientational transformations.

22. The method of Claim 21, wherein the transformations are applied to transformations in related elements.

23. The method of Claim 17, wherein a representation of an element is dependent on a state of the element.

24. The method of Claim 20, wherein an element to be displayed carries the location of the representation and displays the representation with respect to a window.

25. The method of Claim 1, wherein the state of at least one element is stored.

26. The method of Claim 1, wherein each of the states is a type selected from a group consisting of passive, active, not-active, delayed, timed, wait, and conditional.

27. An apparatus for an automated physical system, comprising:
a plurality of elements, each element expressed as a computer implemented object associated with a part of the physical system;
a hierarchy in which the elements are arranged;
wherein there is a plurality of states defined for each element, and the states are aggregated for all of the plurality of elements, thereby to determine a state of the physical system.

28. The apparatus of Claim 27, wherein at least some of the elements are associated with physical devices that are not computer implemented.

29. The apparatus of Claim 27, wherein a plurality of lower level of states are aggregated to determine each of the states.

30. The apparatus of Claim 28, further comprising an associated physical device coupled to some of the elements by a computer implemented device driver.

31. The apparatus of Claim 30, further comprising input or output circuitry coupled between the device driver and the associated physical device.

32. The apparatus of Claim 30, further comprising an interface database coupled to the device driver.

33. The apparatus of Claim 27, wherein the aggregating is performed in one of a simulation mode or a control mode.

34. The apparatus of Claim 27, wherein the aggregating comprises adding or multiplying.

35. The apparatus of Claim 27, wherein the aggregating the states is performed on the results of functional transformation of those states.

36. The apparatus of Claim 27, wherein one hierarchy of the elements includes a plurality of nodes and at least one endnode having no subordinate nodes.

37. The apparatus of Claim 27, wherein the states initiate the transmission of commands.

38. The apparatus of Claim 27, wherein one of the elements is commanded to enter a different state.

39. The apparatus of Claim 27, wherein a state is a representation of an analog or digital value.

40. The apparatus of Claim 27, wherein a structural location of each element is reported.

41. The apparatus of Claim 27, wherein at least one selected element is coupled to other elements to perform a predetermined task of the system.

42. The apparatus of Claim 38, wherein the commanding initiates operation of the system so as to change state of the elements.

43. The apparatus of Claim 27, further comprising:

a user interface displaying each of the elements and the state of each element, and wherein the displayed user interface is periodically updated to illustrate operation of the system.

44. The apparatus of Claim 37, whereas the transmission is dependent on at least one condition.

45. The apparatus of Claim 43, wherein selection of an element opens that element to display its content.

46. The apparatus of Claim 43, further comprising:

a library of representations of elements; and

means for looking up the elements representations in the library.

47. The apparatus of Claim 46, wherein at least some of the elements in the library are related by positional and orientational transformations.

48. The apparatus of Claim 47, wherein the transformations are applied to transformations in related elements.

49. The apparatus of Claim 43, wherein a representation of an element is dependent on a state of the element.

50. The apparatus of Claim 46, wherein an element to be displayed carries the location of the representation and displays the representation with respect to a window.

51. The apparatus of Claim 27, wherein the state of at least one element is stored.

52. The apparatus of Claim 27, wherein each of the states is a type selected from a group consisting of passive, active, not-active, delayed, timed, wait, and conditional.

53. The apparatus of Claim 27, wherein the apparatus is implemented on a first computer, and further comprising a second computer coupled to the first computer, wherein a user accesses the apparatus by the second computer.

54. The apparatus of Claim 27, wherein the apparatus is implemented on a plurality of computers, each computer controlling a portion of the hierarchy of elements.

55. The apparatus of Claim 27, wherein an element which is numerical is a compiled function.

56. The apparatus of Claim 55, further comprising an interface database coupled to the hierarchy, the interface database being associated with the physical system, wherein the function is an entry in the interface database.

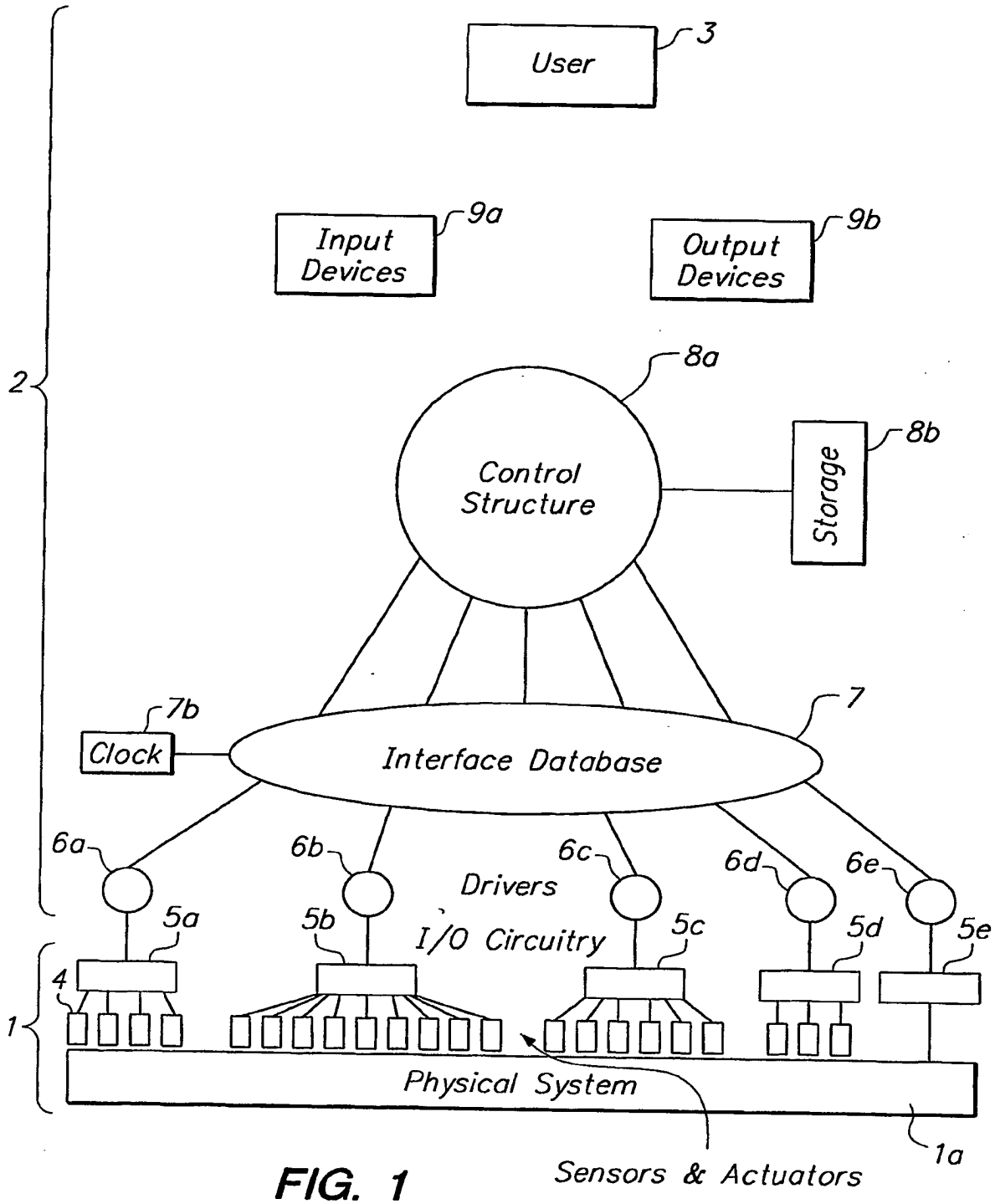


FIG. 1

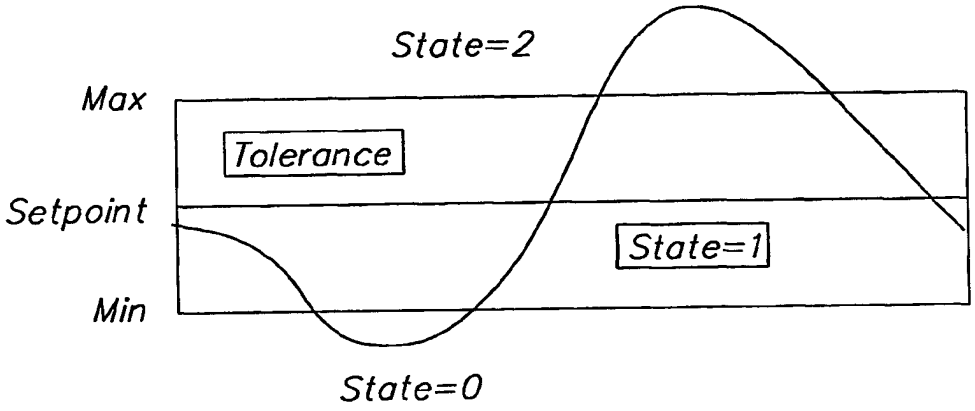


FIG. 2A

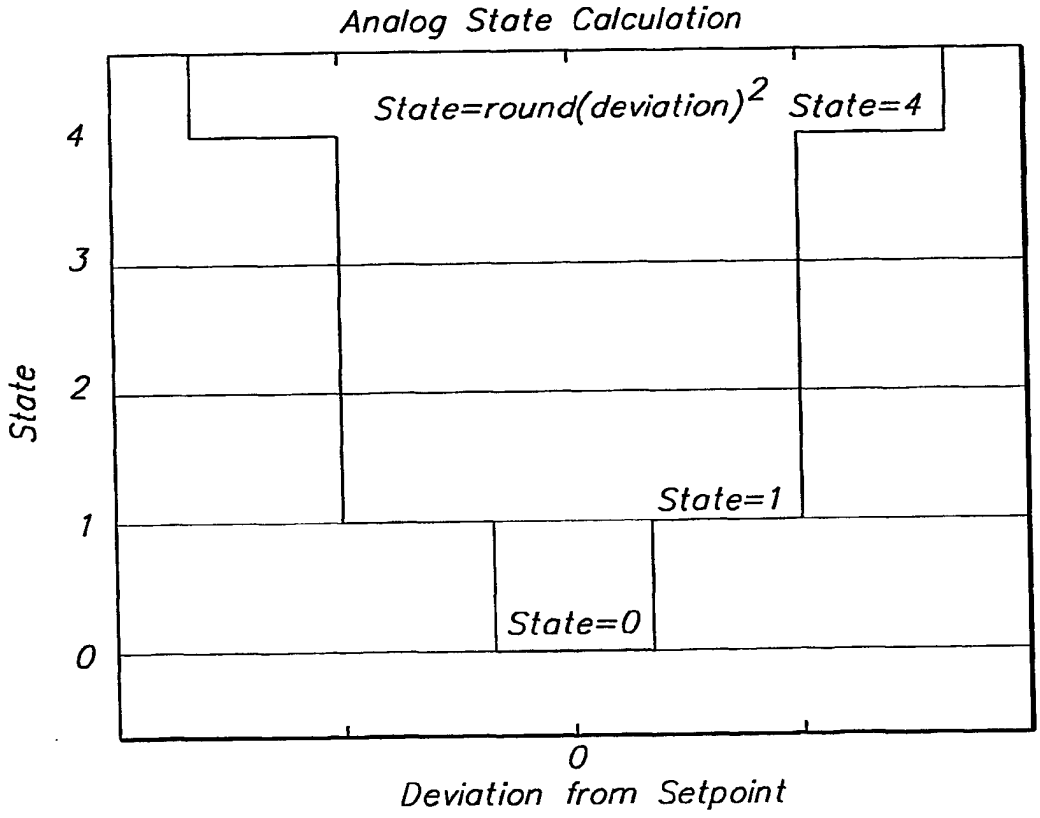


FIG. 2B

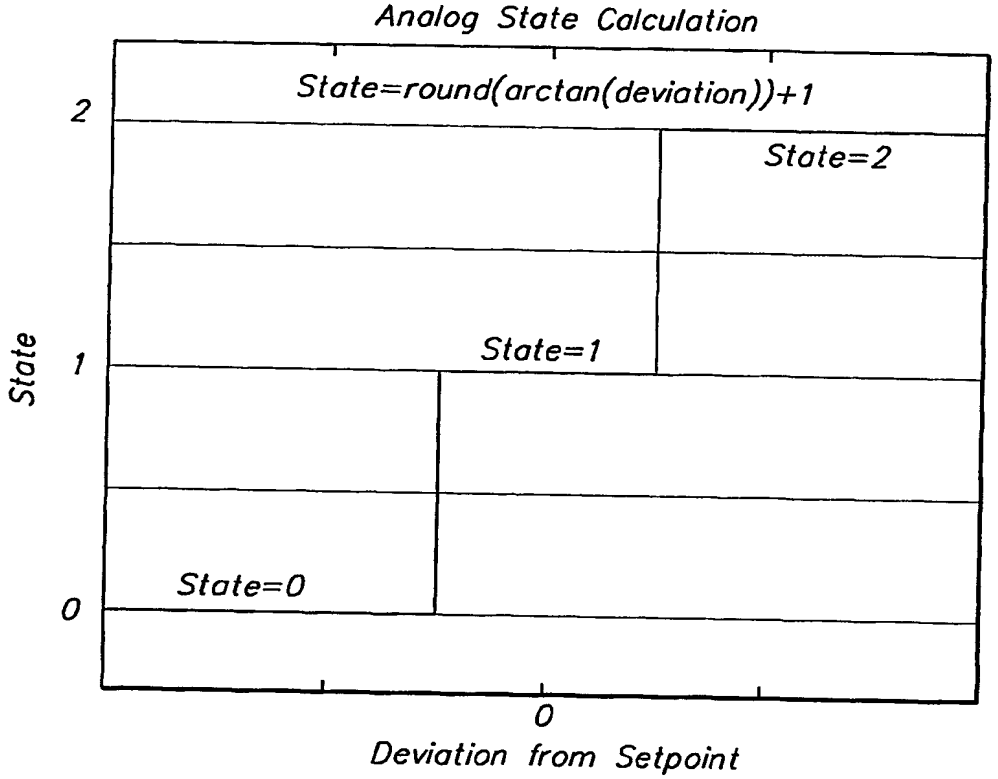


FIG. 2C

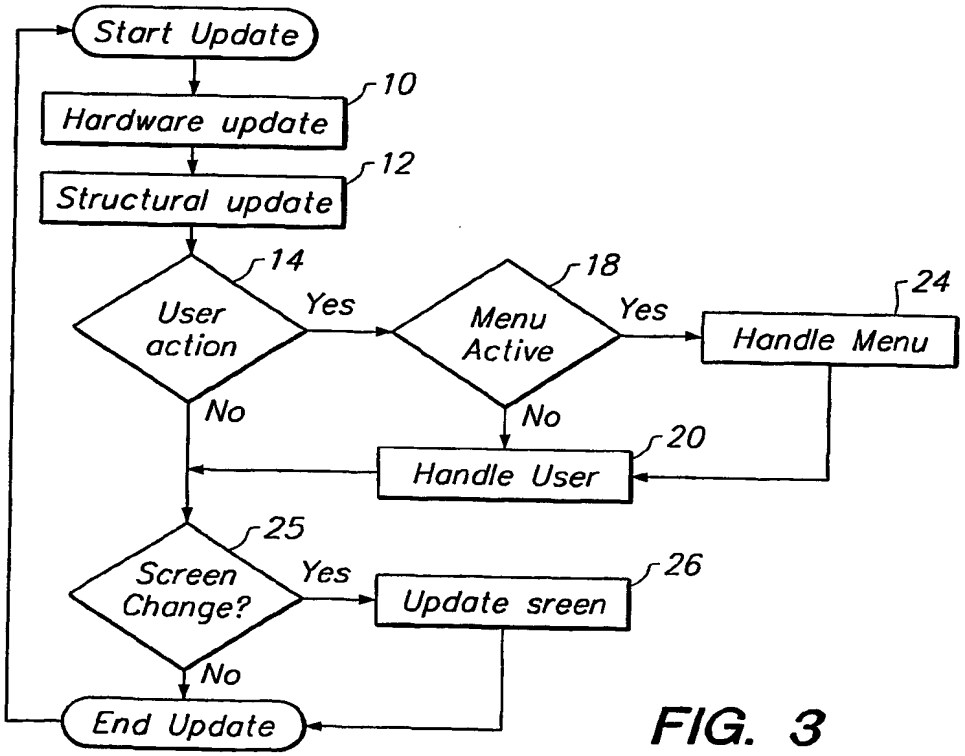


FIG. 3

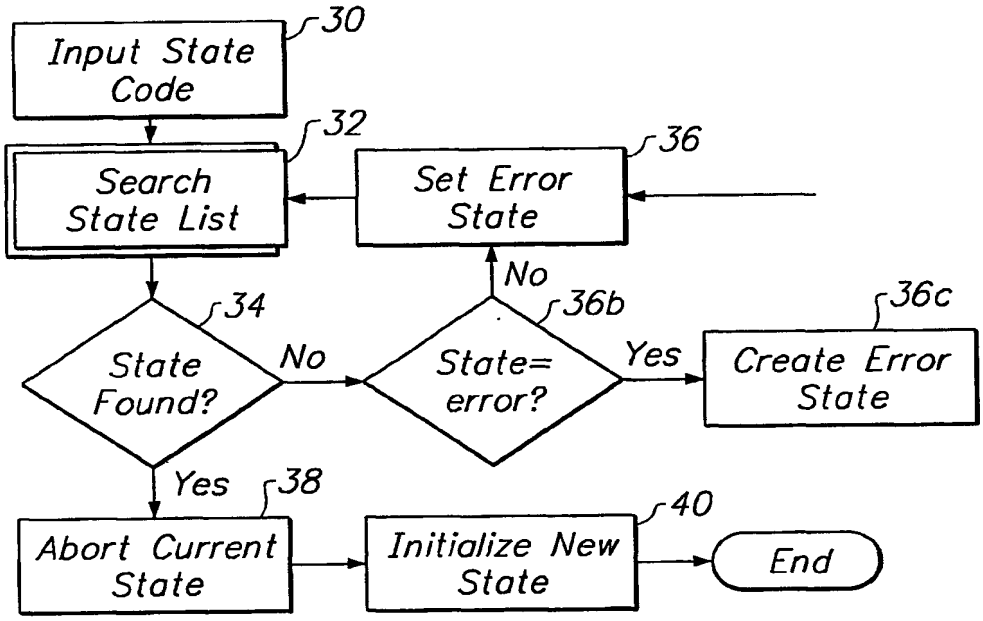


FIG. 4

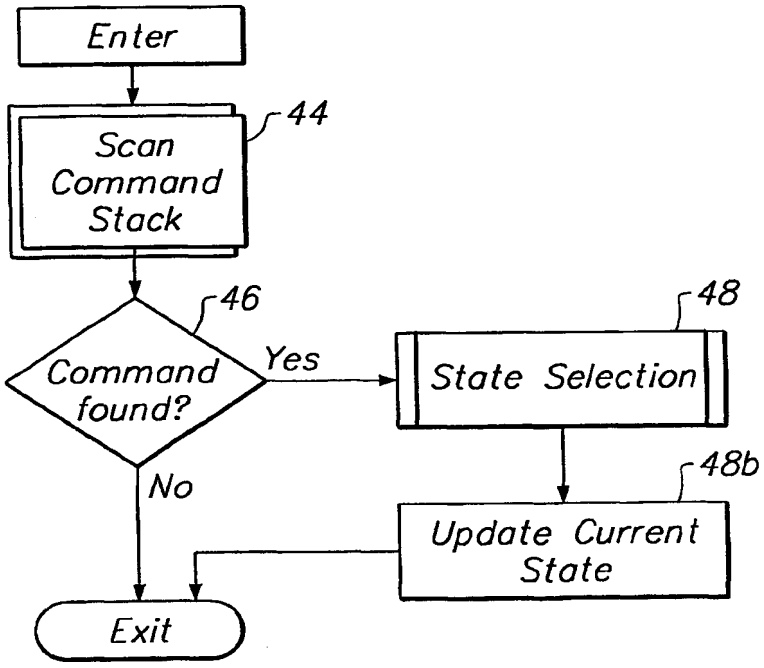


FIG. 5A

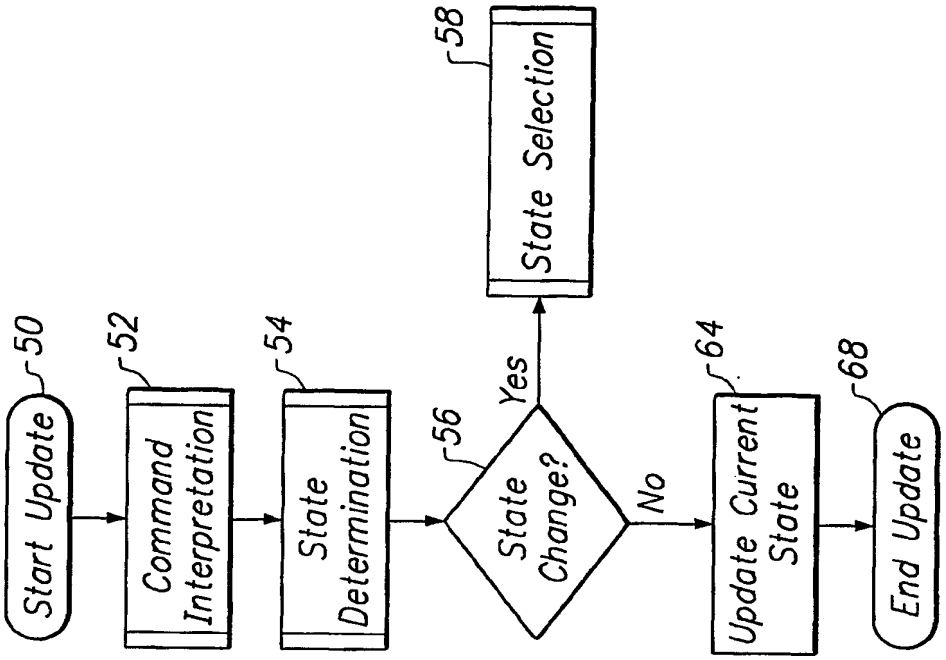


FIG. 6

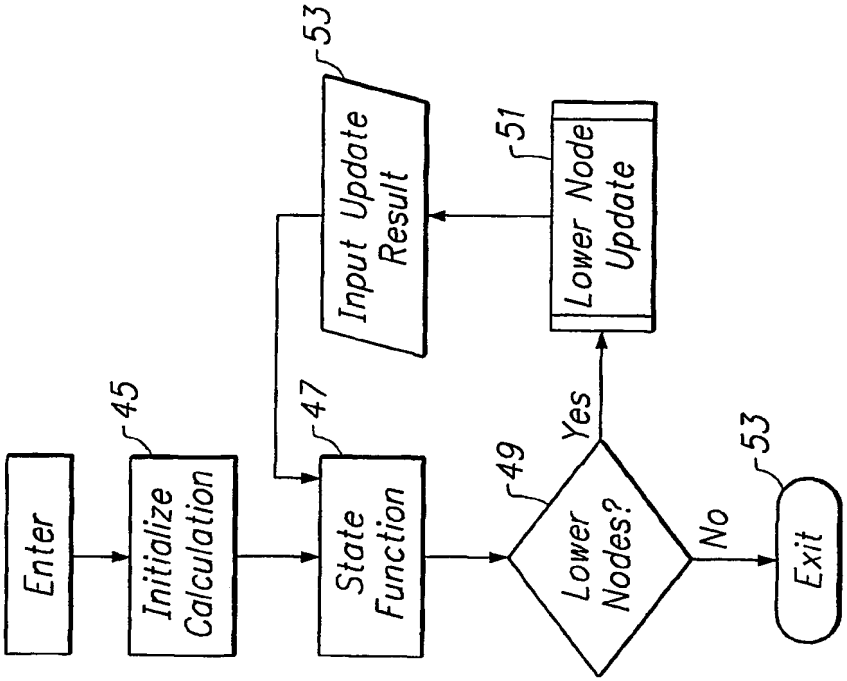


FIG. 5B

```
FUNCTION Update_Code: Number;
    :
Initialize:
    :
IF Command_Found(Command);
    THEN
        Current_State:=Select_State(Command);
        Update_State(Current_State)
    ENDIF;
    :
New_State:=0;
FOR i := 1 TO Number_Of_ChildNodes DO
    New_State:=New_State+ChildNode. Update_Code;
    :
IF New_State<>Current_State
    THEN Current_State:=Select_State(New_State);
Update_State(Current_State);
    :
ENDFUNCTION;
```

FIG. 7

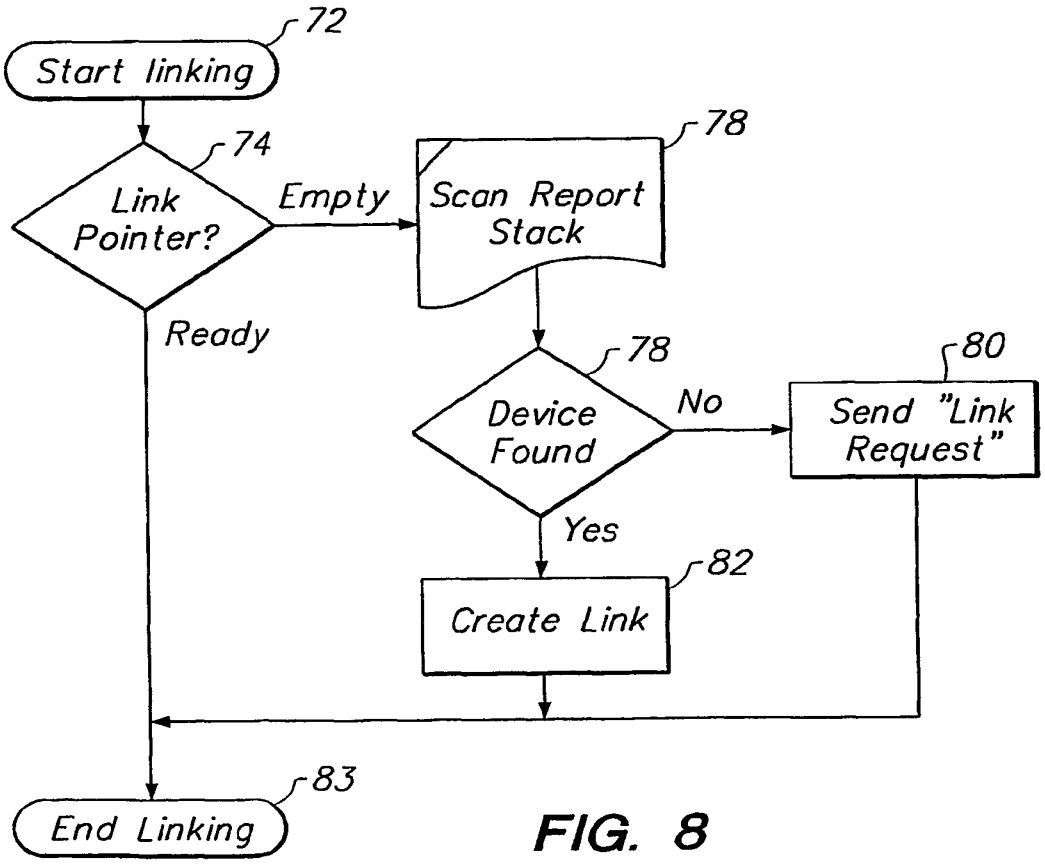


FIG. 8

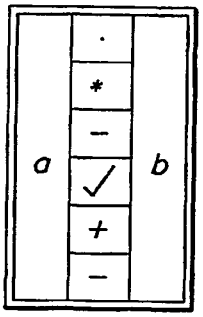


FIG. 9A

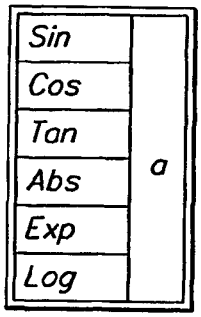


FIG. 9B

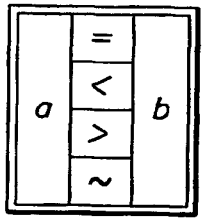


FIG. 9C

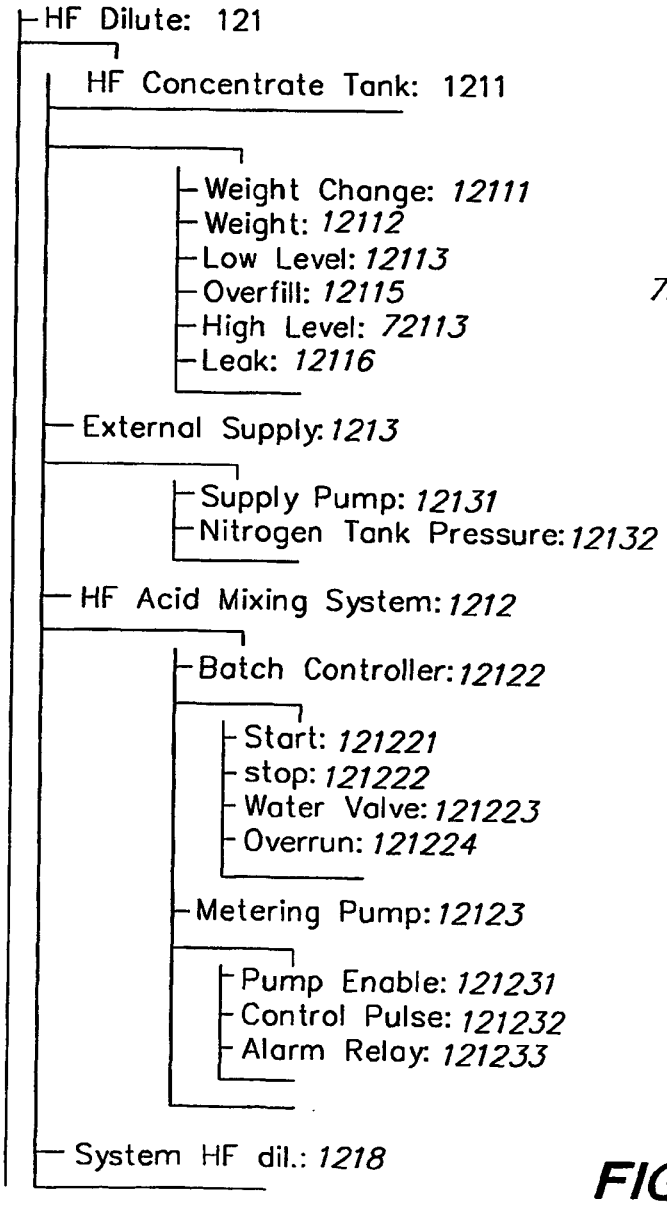


FIG. 12

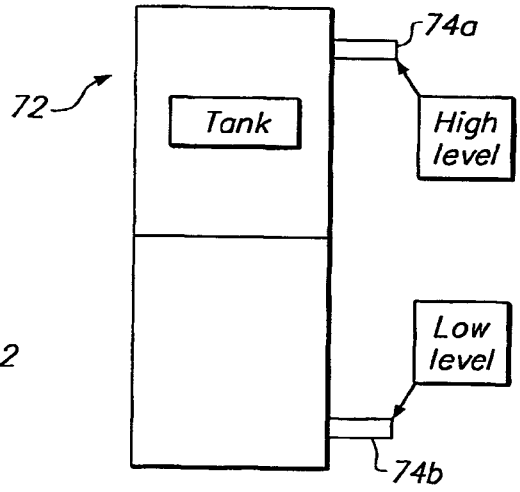


FIG. 10

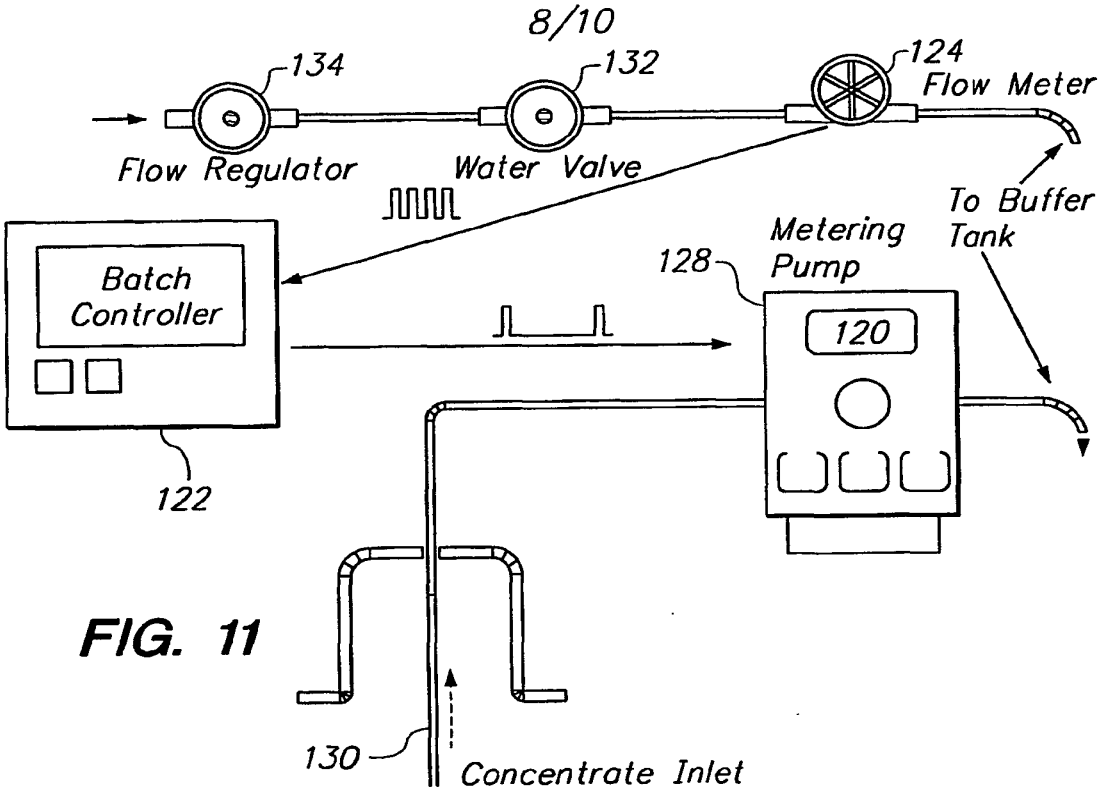


FIG. 11

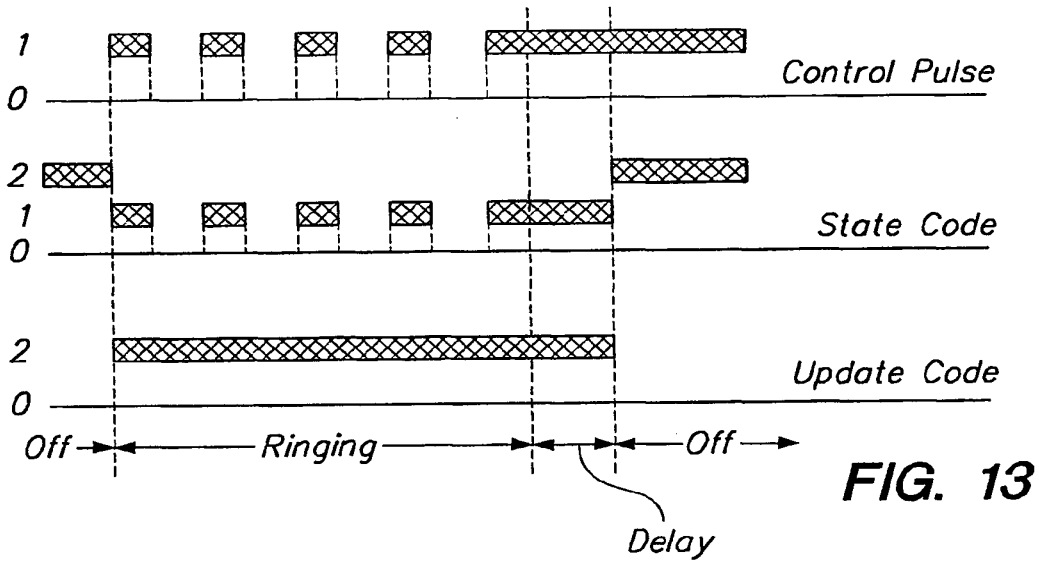


FIG. 13

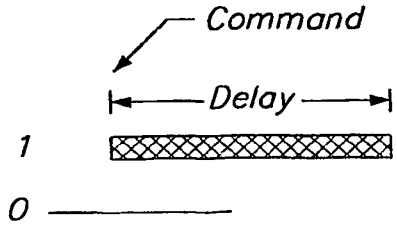


FIG. 14

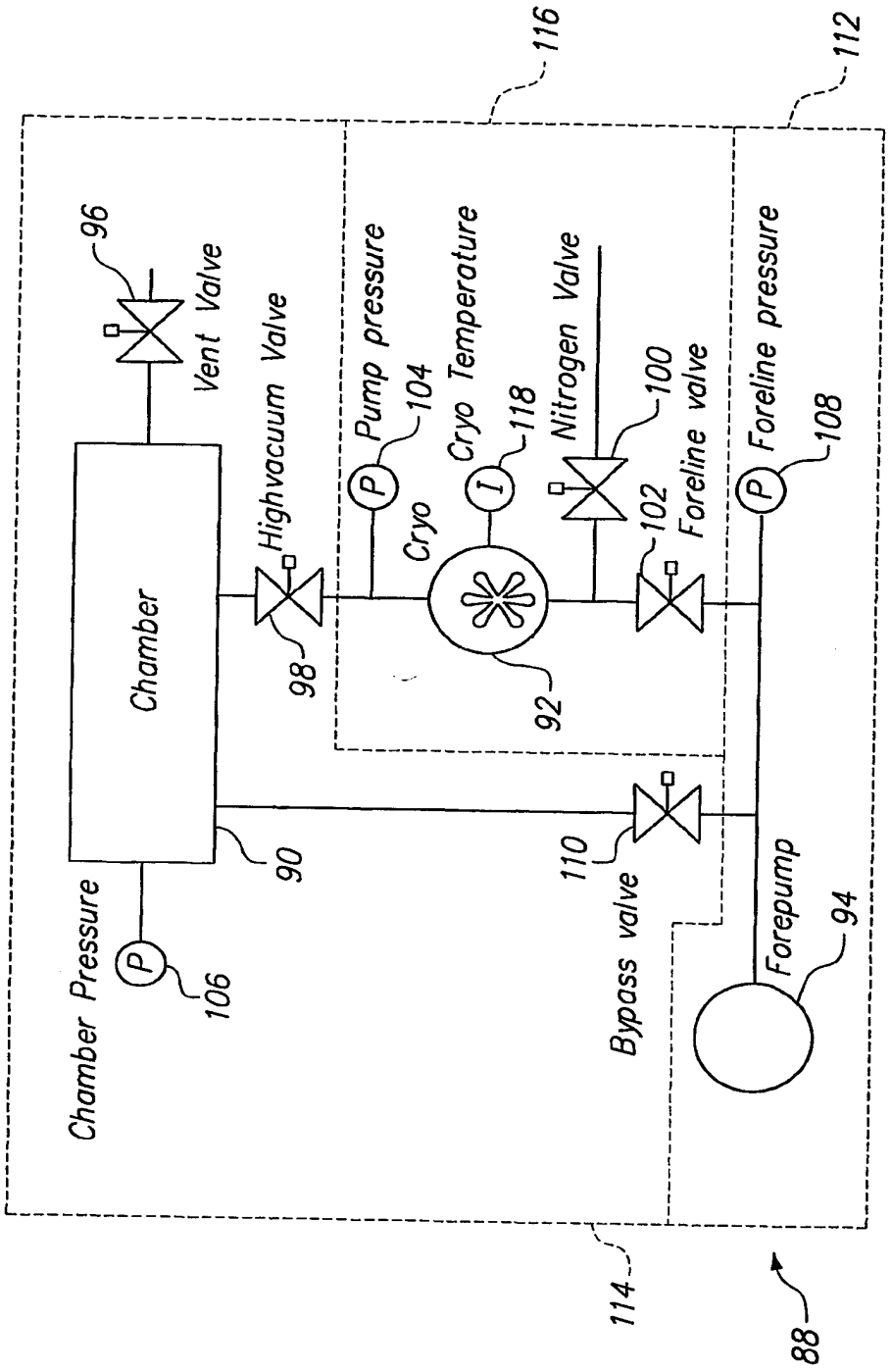


FIG. 15

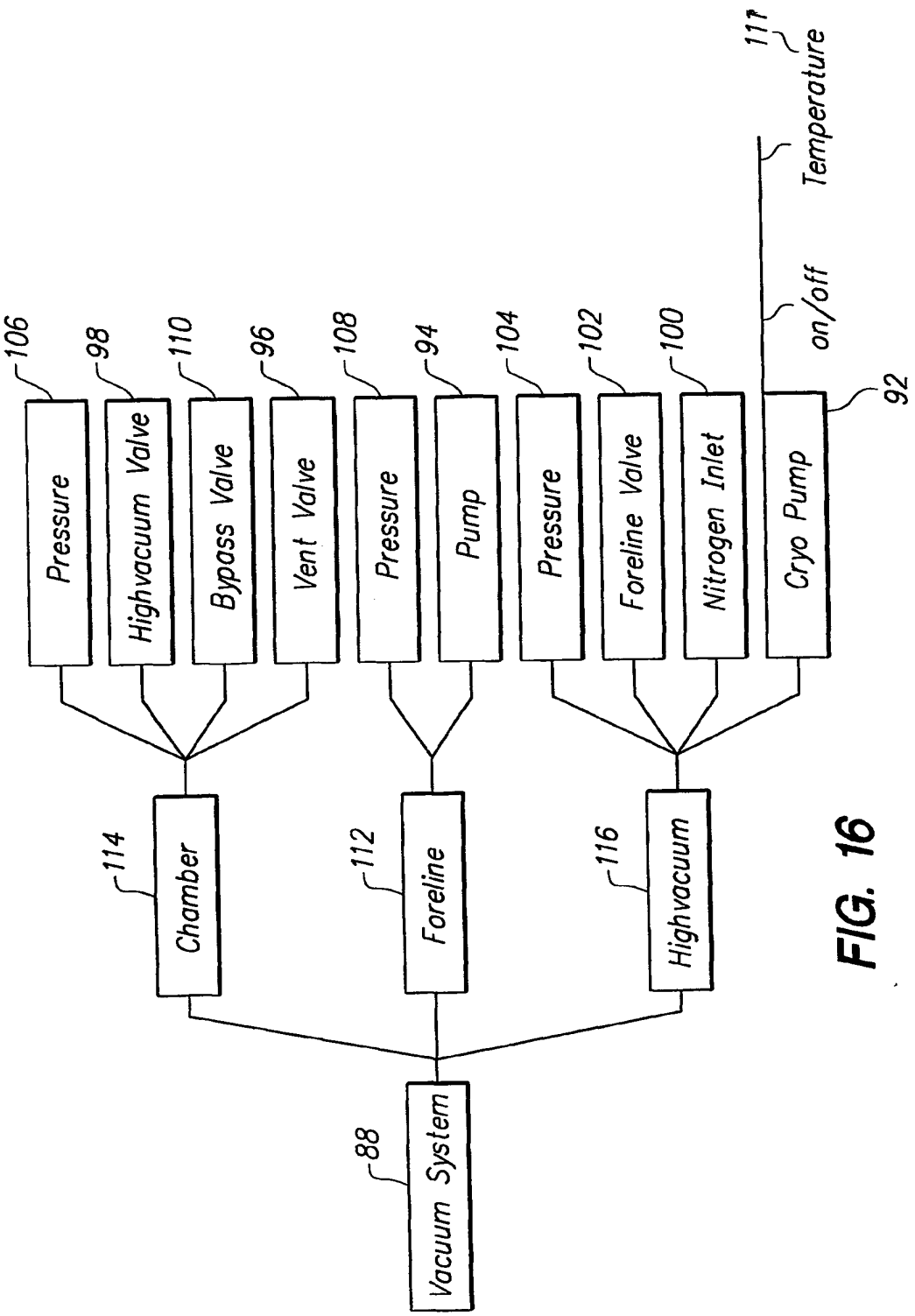


FIG. 16

INTERNATIONAL SEARCH REPORT

Interr ıl Application No
PCT/US 00/18179

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G05B17/02

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G05B

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 97 12301 A (FISCHER HORST ;LOEWEN ULRICH (DE); SONST HORST (DE); FREITAG HARTM) 3 April 1997 (1997-04-03) page 4, line 1 -page 7, line 29	1,7,27, 33
X	US 4 965 743 A (BASHAM BRYAN D ET AL) 23 October 1990 (1990-10-23) column 12, line 1 -column 28, line 49	1,27
X	FR 2 724 744 A (ASS POUR LE DEV DE L ENSEIGNEM) 22 March 1996 (1996-03-22) figure 2	1,27
A	EP 0 482 523 A (OSAKA GAS CO LTD ;UNIV VANDERBILT (US)) 29 April 1992 (1992-04-29)	

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
3 October 2000	11/10/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Kelperis, K

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Intern ... l Application No
PCT/US 00/18179

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 812 394 A (LEWIS ROBERT W ET AL) 22 September 1998 (1998-09-22)	

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 00/18179

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9712301 A	03-04-1997	DE 19639424 A EP 0852759 A	27-03-1997 15-07-1998
US 4965743 A	23-10-1990	NONE	
FR 2724744 A	22-03-1996	NONE	
EP 0482523 A	29-04-1992	JP 6266727 A US 5420977 A	22-09-1994 30-05-1995
US 5812394 A	22-09-1998	NONE	

Form PCT/ISA/210 (patent family annex) (July 1992)